

ARBEITSBERICHTE DES INSTITUTS FÜR MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG (INFORMATIK)

FRIEDRICH ALEXANDER UNIVERSITÄT ERLANGEN NÜRNBERG

HERAUSGEBER : H. Billing, W. Händler, U. Herzog,
F. Hofmann, K. Leeb, P. Mertens, H. Müller, G. Nees,
H. Niemann, D. Seitzer, H. J. Schneider, H. Wedekind

INFERENZ UND ANALYSE SPEZIELLER GRAPHGRAMMATIKEN FÜR DIE SYNTAKTISCHE MUSTERERKENNUNG

Brigitte Bartsch

Dissertation

BAND 13 · NUMMER 6 · ERLANGEN · JULI 1980

1980 № 4 19



ISSN 0344 - 3515



Digitized by the Internet Archive
in 2026

ARBEITSBERICHTE DES INSTITUTS FÜR MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG (INFORMATIK)

FRIEDRICH ALEXANDER UNIVERSITÄT ERLANGEN NÜRNBERG

HERAUSGEBER : H. Billing, W. Händler, U. Herzog,
F. Hofmann, K. Leeb, P. Mertens, H. Müller, G. Nees,
H. Niemann, D. Seitzer, H. J. Schneider, H. Wedekind

INFERENZ UND ANALYSE SPEZIELLER GRAPHGRAMMATIKEN FÜR DIE SYNTAKTISCHE MUSTERERKENNUNG

Der Technischen Fakultät
der Universität Erlangen-Nürnberg
zur Erlangung des Grades
DOKTOR - INGENIEUR

vorgelegt von
Brigitte Bartsch

Erlangen 1980

Dissertation

BAND 13 · NUMMER 6 · ERLANGEN · JULI 1980



Als Dissertation genehmigt
von der Technischen Fakultät
der Universität Erlangen-Nürnberg

Tag der Einreichung: 6.5.1980

Tag der Promotion: 18.7.1980

Dekan: Prof. Dr. H. Niemann

Berichterstatter: Prof. Dr. H. Niemann

Prof. Dr. H.J. Schneider

ISSN 0344-3515

KURZFASSUNG

In dieser Arbeit wird ein neuer Ansatz zur Verwendung von Graphgrammatiken für die syntaktische Mustererkennung vorgestellt.

Der Ansatz besteht aus zwei Varianten eines Inferenzverfahrens zur Gewinnung von Graphgrammatiken mit ketten- und sternstrukturierten Produktionen für Stichproben von Mustern, die durch gerichtete, knoten- und kantenmarkierte Graphen beschrieben werden, und einem darauf abgestimmten Syntaxanalysealgorithmus.

Anwendungen des Ansatzes auf verschiedene Beispielstichproben geben einen Überblick über die Auswirkungen der vorhandenen Möglichkeiten zur Steuerung des Inferenzprozesses.

Erweiterungen des Ansatzes auf programmierte und stochastische Graphgrammatiken werden geschildert und die dadurch erreichbaren Verbesserungen durch Beispiele illustriert.

Zuletzt werden verschiedene Gütekriterien zur Beurteilung von strukturellen und statistischen Eigenschaften der inferierten Graphgrammatiken vorgeschlagen und erprobt.

ABSTRACT

This report presents a new approach to the use of graph grammars for syntactic pattern recognition.

The approach consists of two different algorithms for the inference of graph grammars with chain- and star-structured productions and a corresponding parsing algorithm as well. The grammars are suited to model sample patterns in form of directed graphs with labelled nodes and edges. Some applications of the approach to different graph samples show the influence of the given means to control the inference process.

Extensions of the approach to programmed and stochastic graph grammars are described and the thereby attainable improvements are illustrated by examples.

Finally different goodness criteria for the examination of structural and statistical properties of the inferred grammars are suggested and tested.

INHALTSVERZEICHNIS

Seite

1. Einleitung	1
2. Grundlagen	14
2.1 Grundlagen zum Bereich Graphen und Graphgrammatiken	14
2.1.1 Graphen	14
2.1.2 Graphgrammatiken	18
2.2 Grundlagen zum Bereich Inferenz von Grammatiken und Syntaxanalyse	24
2.2.1 Inferenz von Grammatiken	24
2.2.2 Syntaxanalyse	28
3. Inferenz von Graphgrammatiken	34
3.1 Allgemeine Vorbemerkungen	34
3.1.1 Vorbemerkungen zur Stichprobe	35
3.1.2 Vorbemerkungen zur Form der Produktionen	41
3.1.3 Vorbemerkungen zur Inferenzstrategie	51
3.2 Inferenzverfahren	54
3.2.1 Inferenz ohne Gedächtnis	54
3.2.1.1 Darstellung und Bearbeitung der Stichprobenelemente	55
3.2.1.2 Konstruktion einer Einzelgrammatik	58
3.2.1.3 Vereinigung der Einzelgrammatiken	64
3.2.1.4 Parametergesteuerte Grammatikkonstruktion	68
3.2.1.5 Weitere Beispiele und Erfahrungen	76
3.2.2 Inferenz mit Gedächtnis	82
3.2.2.1 Konstruktion von Einzelgrammatiken unter Verwendung der aktuellen Zwischengrammatik	83
3.2.2.2 Weitere Beispiele und Erfahrungen	91
4. Syntaxanalyse der Graphgrammatiken	94
4.1 Allgemeine Vorbemerkungen	94

4.2	Syntaxanalysealgorithmus	97
4.3	Weitere Beispiele und Erfahrungen	101
5.	Erweiterung des Ansatzes auf programmierte Graphgrammatiken	104
5.1	Inferenz von programmierten Graphgrammatiken	104
5.2	Syntaxanalyse mit programmierten Graphgrammatiken ..	110
6.	Erweiterungsmöglichkeiten des Ansatzes auf stochastische Graphgrammatiken und stochastische Programmierung	116
6.1	Inferenz von stochastischen Graphgrammatiken	116
6.2	Syntaxanalyse von Graphgrammatiken mit stochastischer Programmierung	123
7.	Gütekriterien zum Vergleich von Graphgrammatiken	127
7.1	Strukturelle Gütekriterien	127
7.2	Statistische Gütekriterien	137
8.	Vorschläge für weitere Untersuchungen	140
	Zusammenfassung	142
	Schlußbemerkung	145
	Literaturverzeichnis	146

1. EINLEITUNG

Die verschiedenen Methoden der Mustererkennung lassen sich im wesentlichen in zwei Gruppen einteilen, die als *numerische Klassifikationsverfahren* und *nichtnumerische Verfahren zur Analyse und Klassifikation* von Mustern [Ni 74] bezeichnet werden.

Klassifikationsverfahren sollen die Aufgabe lösen, eine Menge von meist relativ einfachen Mustern in eine Reihe von Musterklassen einzuordnen. *Analyseverfahren* sollen die für die weitere Verarbeitung erforderliche Information über komplexere Muster z.B. in Form von einfacheren Bestandteilen und deren Beziehungen untereinander liefern.

Der *numerische* Ansatz geht davon aus, daß es möglich ist, die für die Klassifikation eines Musters benötigte Information auf einen Vektor von *Merkmale*n in Form von gemessenen oder aus Messungen berechneten Zahlenwerten abzubilden, und daß die Kenntnis eines Klassennamens für die weitere Verarbeitung eines Musters ausreicht. Die Klassifikation selbst erfolgt dann allein auf der Basis der im Merkmalsvektor enthaltenen Information z.B. durch Aufteilung des Merkmalsraums in Bereiche, die jeweils einer Klasse zugeordnet werden können.

Die Voraussetzungen für den Einsatz numerischer Klassifikationsverfahren sind für viele Arten von Mustern wie z.B. Schriftzeichen, gesprochene Einzelworte, Werkstücke, einzelne Blutzellen u.a.m. durchaus erfüllt; daneben gibt es jedoch eine wachsende Zahl von Mustererkennungsaufgaben wie z.B. Analyse von Luftbildern oder Blasenkameraaufnahmen, Auswertung von Röntgenbildern und ähnlichem medizinischen Bildmaterial, bei denen die Struktur der Muster eine ganz wesentliche Rolle spielt. Hier ist es oft schwierig, eine der Problemstellung angemessene Einteilung in Musterklassen vorzunehmen und die weitere Verarbeitung allein durch einen Klassennamen zu steuern. Außerdem würde eine Vorgehensweise wie bei den numerischen Verfahren wegen der im allgemeinen größeren Komplexität der Muster nur mit entsprechend höherem

Aufwand oder wegen der unzureichenden Erfassbarkeit von Struktureigenschaften vielleicht überhaupt nicht zum Ziel führen.

Deshalb werden für die Verarbeitung komplexer Muster hauptsächlich *nichtnumerische Methoden* (auch *syntaktische*, *linguistische* oder *strukturelle Methoden* genannt) vorgeschlagen und angewendet. Die Grundidee hierbei ist, ein Muster in einfacher strukturierte Teilmuster (*Grundsymbole*) zu zerlegen und den Aufbau des komplexen Musters aus Grundsymbolen und deren Beziehungen untereinander (*Grundrelationen*) in einer *Bildbeschreibungssprache* auszudrücken.

Diese Vorgehensweise besitzt große Ähnlichkeit mit dem Aufbau von Sätzen einer natürlichen Sprache aus Wörtern bzw. Buchstaben. Aus diesem Grunde ist es naheliegend, die Vorschriften über den Aufbau der Sätze einer Bildbeschreibungssprache durch Regeln einer *Grammatik* darzustellen und eine bestimmte Grammatik als *Modell* für die von ihr erzeugbare Menge von Musterbeschreibungen anzusehen. Die Klassifikation eines Musters erfolgt dann auf der Basis der in der Musterbeschreibung enthaltenen Information z.B. durch die Feststellung, welche Grammatik die Musterbeschreibung erzeugen kann, welche Produktionen dabei verwendet werden o.ä.

Für das Auffinden und Auswählen geeigneter Grundsymbole und Grundrelationen für einen bestimmten Problemkreis gibt es bisher noch keinen allgemein anwendbaren Lösungsweg, aber genügend brauchbare Ansätze und Beispiele dafür, wie man z.B. gekrümmte Linien durch gerade oder gekrümmte Liniensegmente approximieren oder Flächen aus Teilflächen zusammensetzen kann [Fu 76a, Pa 77], um das Problem der Festlegung, aus welchen Grundsymbolen und Grundrelationen die Muster eines Problemkreises aufbaubar sein sollen, im folgenden als einigermaßen zufriedenstellend lösbar voraussetzen zu können.

Die Wahl der "richtigen" Grundsymbole und Grundrelationen ist von großer Bedeutung für die Komplexität der Erstellung der Musterbeschreibungen und der dazugehörigen Grammatik. Läßt sich beispielsweise ein Muster durch seine Umrißform ausreichend charakterisieren, so ist es zweckmäßig, hierfür nach einer Beschreibung zu suchen, die den Umriß des Musters durch

eine Aneinanderreihung von Umrißlinienssegmenten darstellt. Jedes dieser Segmente ist durch ein Zeichen repräsentierbar, so daß die Beschreibung einzelner Muster in Form einer Zeichenkette und die Beschreibung einer Menge von Mustern durch eine Zeichenkettengrammatik möglich ist. Beispiele für die erfolgreiche Anwendung dieser Vorgehensweise sind die bekannte Chromosomengrammatik nach Ledley et al. [Le 65] oder Grammatiken zur Erzeugung von einfachen geometrischen Figuren wie Dreiecken, Quadraten u.ä. [Fu 74].

Falls aber ein Muster neben einer Konturlinie noch weitere wichtige Strukturelemente besitzt, kommt man mit einer linearen Aneinanderreihung von Grundsymbolen nicht mehr aus und muß einen Bildbeschreibungsformalismus entwickeln, der die Verknüpfung von mehr als zwei Grundsymbolen in einem "Verknüpfungspunkt" zuläßt. Beispiele hierfür sind Shaws picture description language PDL [Sh 69] und Feders plex language [Fe 71]. Beide Ansätze lösen das Problem der Notation von ursprünglich zweidimensionalen Zusammenhängen in eindimensionaler Form durch Einführung von Klammern und Verknüpfungsregeln, wodurch bei komplexeren Mustern die Beschreibungen sehr schnell unübersichtlich werden und dadurch schwierig zu handhaben und zu kontrollieren sind. Außerdem gibt es bei diesen eindimensionalen Bildbeschreibungssprachen für ein Muster in der Regel mehrere zulässige Beschreibungen, deren Äquivalenz meistens nicht auf den ersten Blick feststellbar ist.

Aus diesen Erfahrungen heraus wurden schon ziemlich bald weitergehende Formalismen entwickelt, die es erlauben, *zweidimensionale Zusammenhänge* auch *zweidimensional* darzustellen.

Die wichtigsten Bildbeschreibungssprachen dieser Kategorie sind die "tree language" (Baumsprache) nach Brainerd [Br 69], die von Fu und Bhargava [FuBh 73] zur Beschreibung von Blasen-kammerspuren erstmalig eingesetzt und später zur Analyse von Fingerabdrücken und Luftbildern angewendet wurde, die "web language" (Gewebesprache) von Pfaltz und Rosenfeld [PfRo 69], die zur Beschreibung von Neuronennetzen und Luftbildern Verwendung fand und die "graph language" (Graphsprache) von Pavlidis [Pa 72], die bisher mehr für theoretische Unter-

suchungen eingesetzt wurde.

Die Unterschiede zwischen den verschiedenen zweidimensionalen Bildbeschreibungsformalismen bestehen hauptsächlich in den unterschiedlichen Möglichkeiten, gerichtete oder ungerichtete, markierte oder nichtmarkierte Kanten zwischen markierten oder nichtmarkierten Knoten darstellen zu können. Bäume (trees) haben markierte Knoten, zwischen denen ungerichtete, nicht-markierte Kanten in einer Baumstruktur angeordnet sind. Gewebe [Wi 78] bzw. Netze [Ni 74] (webs) bestehen nach Pfaltz/Rosenfeld aus markierten Knoten, zwischen denen gerichtete, unmarkierte Kanten in beliebiger Anordnung möglich sind. Von Brayer und Fu [BrFu 75] existiert eine Arbeit über die Verwendung von webs mit Knoten- und Kantenmarkierungen z.B. für die Darstellung von Wolke-Schatten-Paaren auf Luftbildern u.ä. Die Graphen (graphs) von Pavlidis bestehen aus nicht-markierten Knoten, zwischen denen nichtmarkierte, ungerichtete Kanten so angeordnet werden dürfen, daß bei Graphen n -ter Ordnung maximal n Kanten in einem Knoten zusammentreffen.

Um diese Vielfalt von Begriffen etwas zu vereinheitlichen, soll im folgenden nur noch zwischen Bäumen und Graphen unterschieden werden, wobei die Graphen durch Attribute wie gerichtet, mit oder ohne Knoten - bzw. Kantenmarkierung etc. näher beschrieben werden können.

In Bezug auf die Darstellbarkeit von beliebigen Musterstrukturen ist das allgemeinste und flexibelste Konzept das der gerichteten Graphen mit sowohl Knoten- als auch Kantenmarkierungen ohne Beschränkung der Anzahl der in einem Knoten ein- bzw. auslaufenden Kanten. Mit diesem Konzept sind auch alle einfacheren Strukturen wie Zeichenketten, Bäume oder Graphen mit nur Knoten- oder nur Kantenmarkierungen darstellbar, indem man z.B. für nicht benötigte Knoten oder Kanten solche mit einheitlicher Markierung einführt oder bei Strukturen mit ungerichteten Kanten entsprechende Doppelkanten vorsieht.

Der Hauptgrund dafür, daß für den größten Teil der bisherigen Anwendungen syntaktischer Methoden in der Mustererkennung die

einfacheren Formalismen wie Beschreibung durch Zeichenketten oder Bäume Verwendung fanden, besteht darin, daß die anschließende Konstruktion von Grammatiken und Verarbeitung weiterer Muster mit Hilfe dieser Grammatiken umso aufwendiger und komplizierter wird, je weniger Restriktionen die Bildbeschreibungssprache enthält. Insbesondere in den Anfangsjahren der syntaktischen Mustererkennung lag es deshalb nahe, sich die vorhandene Theorie der formalen Zeichenkettensprachen [Ch 59] und die dafür entwickelten Syntaxanalysealgorithmen [AhUl 72] zunutze zu machen und zuerst einmal die Möglichkeiten struktureller Bildbeschreibungen auf Zeichenkettenbasis auszuloten [Fu 74].

Bei der Entwicklung syntaktischer Mustererkennungsverfahren zeigte sich ziemlich bald, daß es bei Vorliegen einer größeren Menge von Musterbeschreibungen zu einem nichttrivialen Problem werden kann, eine geeignete *Grammatik als eine Art Konstruktionsvorschrift für den Aufbau der Muster* anzugeben. So sind Grammatiken für kleinere Mengen von Zeichenketten noch ganz gut "von Hand" konstruierbar, aber für Stichproben von z.B. tausend Bäumen zur Darstellung von Fingerabdrücken oder Blasenkammerereignissen sind automatische Verfahren unabdingbar. Deshalb interessiert man sich seit circa zehn Jahren von der Seite der Mustererkennung her dafür, ob und mit welchem Erfolg es möglich ist, die Aufgabe des Findens einer Grammatik, die eine vorgegebene Menge von Mustern erzeugt, einem Computer zu übertragen.

Die ersten mehr linguistisch orientierten Gedanken zu diesem Problem der *Inferenz von Grammatiken* (grammatical inference) gehen zurück auf Chomsky [Ch 64, zitiert nach Cr 70]:

"Consider the abstract problem of constructing an 'acquisition model' for language, that is, a theory of language learning or grammar construction. Clearly, a child who has learned a language has developed an internal representation of a system of rules that determines how sentences are to be formed, used and understood.We can say that the child has developed and internally represented a generative grammar.... He has done this on the basis of observation of what we may call primary linguistic data. This must include ... well-formed

sentences, and may include also examples designated as non-sentences, and no doubt much other information of the sort that is required for language learning ... On the basis of such data the child constructs a grammar — that is, a theory of the language of which the well-formed sentences of the primary linguistic data constitute a small sample".

Seine Ausführungen enthalten bereits eine Reihe von Vorstellungen, die sich in späteren Arbeiten auf diesem Gebiet wiederfinden [Go 67, Wi 70]. Die ersten mathematisch formulierten Überlegungen zum Inferenz-Problem stammen von Solomonoff [So 64], der im Rahmen einer umfangreicheren Arbeit über induktives Schließen auch die Frage nach Möglichkeiten, eine Menge von Zeichenketten mit Hilfe einer Chomsky-Grammatik zu "extrapolieren" aufwirft und Lösungsansätze durch Aufzählung von Grammatiken bzw. Umformung von Grammatiken mit Hilfe eines Optimierungskriteriums vorschlägt, die später von Horning [Ho 69] bzw. Cook [CoRoAr 76] aufgegriffen und realisiert wurden. Gold [Go 67] beschäftigte sich als erster mit den theoretischen Randbedingungen, unter denen eine Grammatik für eine bekannte Sprache "erlernbar" ist. Weitere Ergebnisse hierzu lieferten Feldman [Fe 72] und Maryanski [Ma 74].

Die ersten auf einem Rechner implementierten Inferenzverfahren entwickelten sich in zwei grundlegend verschiedene Richtungen. Die sogenannten *enumerativen Verfahren* basieren auf der Tatsache, daß sich Grammatiken einer vorgewählten Form über einem gegebenen terminalen Alphabet von Grundsymbolen und Grundrelationen systematisch aufzählen lassen. Die Grundidee dieser Gruppe von Verfahren besteht darin, solange Kandidaten-Grammatiken aufzuzählen und auf ihre Eignung zur Erzeugung der Stichprobe zu testen, bis eine Grammatik den gestellten Anforderungen genügt und als Lösungsgrammatik akzeptiert wird. Der größte Vorteil der enumerativen Verfahren ist ihre Eigenschaft, Grammatiken liefern zu können, die in bezug auf ein selbstgewähltes Kriterium optimal sind. Der große Nachteil dieser Verfahren besteht darin, daß man für die erschöpfende Suche nach der besten Grammatik mit einem enorm hohen Aufwand an Rechenzeit bezahlen muß.

Die bekanntesten Verfahren dieser Richtung stammen von Horning

[Ho 69], Maryanski [MaBo 77] und Wharton [Wh 77]. Sie alle brauchen für die Inferenz von regulären oder kontextfreien Zeichenkettengrammatiken mit wenigen nichtterminalen und terminalen Symbolen noch soviel Rechenzeit, daß es trotz aller Erfolge bei der Aufwandsreduzierung enumerativer Inferenzalgorithmen und trotz der enormen Fortschritte bei der Leistungssteigerung von Rechenanlagen bis heute nicht empfehlenswert ist, umfangreichere praktische Probleme aus der Mustererkennung mit einem enumerativen Inferenzverfahren anzugehen [MuWa 78, El 79].

Der andere Ansatz, die Gruppe der *konstruktiven Verfahren*, geht zurück auf Algorithmen zur Synthese von Automaten auf der Basis ihres Ausgabeverhaltens. Diese Verfahren entwickeln i.a. zunächst eine triviale Grammatik, die genau die Stichprobe modelliert, und versuchen diese meist recht umfangreiche Grammatik durch Auffinden von Gemeinsamkeiten und Ähnlichkeiten im Erzeugungsprozeß der einzelnen Stichprobenelemente zu komprimieren und zu verallgemeinern. Die Regeln, nach denen die Umformung der Grammatik vorgenommen wird, sind teils theoretisch begründbarer, teils heuristischer Natur und variieren auch abhängig von der Form der Produktionen von Verfahren zu Verfahren ziemlich stark.

Zu den ersten eine Stichprobe verallgemeinernden konstruktiven Inferenzverfahren gehört die nach Vorarbeiten von Feldman [FeGiHoRe 69] von Biermann und Feldman [BiFe 70] entwickelte k-tail-Methode zur Inferenz von regulären Zeichenkettengrammatiken. Die Idee hierbei ist, die Erzeugung von Stichprobenelementen mit gleichen "Satzanfängen" bzw. "Satzenden" zusammenzufassen, wobei die Länge übereinstimmender Satzenden Einfluß auf die Menge der zusätzlich zur Stichprobe erzeugbaren Zeichenketten hat und mittels eines Parameters variiert werden kann.

In der Folgezeit wurde versucht, den konstruktiven Ansatz auf größere Klassen von Zeichenkettengrammatiken hin zu erweitern. Beiträge hierzu leisteten Gips [FeGiHoRe 69] mit einem Inferenz-Algorithmus für Pivot-Grammatiken und Crespi-Reghizzi mit Verfahren zur Gewinnung von Operator-Präzedenz-Grammatiken

[Cr 71a] und k -verschiedenen, k -homogenen Grammatiken [Cr 71b]. Diese drei Grammatiktypen erzeugen eingeschränkt kontextfreie Sprachen, die sich wegen einiger Voraussetzungen über den Aufbau der Stichprobenelemente nur in Spezialfällen zur Beschreibung von Musterstrukturen eignen. Trotzdem ist das Verfahren aus [Cr 71b] mit Erfolg zur Chromosomenklassifikation [LeFu 74] eingesetzt worden. Uneingeschränkt kontextfreie Grammatiken liefert das Inferenzverfahren von Cook [CoRo 76], bei dem die Anwendung verschiedener Umformungsmöglichkeiten der Grammatik durch eine Kostenfunktion gesteuert wird und das mit relativ geringem Rechenaufwand nicht immer optimale, aber meistens ganz brauchbare Ergebnisse bringt [Du 79]. Beachtenswert ist in diesem Zusammenhang auch ein Ansatz von Evans [Ev 71] zur Gewinnung einer speziellen Sorte kontextfreier Bildbeschreibungsgrammatiken.

Neben den erwähnten Inferenzverfahren sind noch etliche weitere Algorithmen zur Inferenz von regulären Grammatiken [Mi 76, Mi 78, Po 76] und ein Vorschlag zur Inferenz von "transition network grammars" (TN-Grammatiken) [ChFu 76] publiziert worden, aber man muß insgesamt konstatieren, daß es bis heute kein praktikables Inferenzverfahren für kontextsensitive Grammatiken gibt und daß das gesamte Gebiet der Inferenz von Zeichenkettengrammatiken in den letzten Jahren etwas zu stagnieren scheint [Fu 78]. Die Ursachen dafür sind sicherlich weniger in der Komplexität des Inferenzproblems für kontextsensitive Grammatiken zu suchen, sondern eher darin, daß die Problemkreise von Mustern, die sich durch Zeichenketten Sprachen adäquat beschreiben lassen, unter den derzeitigen Hauptanwendungsgebieten der Mustererkennung [Ni 30a] nicht sehr zahlreich vertreten sind [Fu 77a] und sich seit etwa fünf Jahren zunehmend die Erkenntnis durchsetzt, daß man für zukünftige Anwendungen der syntaktischen Mustererkennung *leistungsfähigere Formalismen als formale Zeichenketten Sprachen braucht* [Fu 74, Ka 74, Pa 77].

Ein erster Schritt zur Inferenz solcher Formalismen waren zwei einander weitgehend ähnliche Inferenzverfahren für expansive Baumgrammatiken [Br 69], die von Bhargava/Fu [BhFu 74] und

Gonzalez/Thomason [GoTh 74] im Jahr 1974 auf der gleichen Tagung vorgestellt wurden. Beide Verfahren generieren für jeden Baum der Stichprobe eine relativ umfangreiche triviale Grammatik und versuchen dann, die Gesamtgrammatik durch Zusammenfassung äquivalenter nichtterminaler Symbole soweit wie möglich zu komprimieren. Verbesserungen und Erweiterungen dieser Verfahren wurden von Brayer/Fu [BrFu 77] bzw. Gonzalez/Thomason [GoTh 78] dahingehend vorgenommen, daß die Voraussetzungen für die Verschmelzung zweier nichtterminaler Symbole über die Eingabe von Parametern verschärft werden können. Beide Verfahren wurden für praktische Probleme wie die Auswertung von Blasenkamerspuren [Bh 74], Fingerabdrücken [MoFu 76], handgeschriebenen Druckbuchstaben [LuFu 78], Luftbildern [Fu 76b], Texturen [LuFu 79] und Umrissen von Fahrzeugen [GoTh 78] mit Erfolg eingesetzt.

Die besten Erfahrungen mit Bäumen als Bildbeschreibungsformalismus macht man bei hierarchisch strukturierten oder generell azyklisch aufgebauten Mustern, deren Verzweigungsstrukturen so komplex sind, daß sie mit Verbindungen zu einem rechten und einem linken Nachbarn wie bei Zeichenketten nur mehr unter Schwierigkeiten darstellbar wären. Probleme mit Baumbeschreibungen können entstehen, wenn die in Bäumen vorhandene Ordnung der Knoten eines Niveaus nicht mit der Strategie der Musterbeschreibung übereinstimmt [Ka 74] oder zyklische Muster nur sehr umständlich oder gar nicht in Baumform darstellbar sind [Ho 80].

Zur Überwindung dieser Probleme bietet sich die Verwendung adäquaterer Formalismen wie Graphen und Graphgrammatiken geradezu zwangsläufig an [Ka 74, BrFu 75, Fu 76a, Pa 77], doch hat diese Situation bisher noch zu keinem realisierten Inferenzverfahren für Graphgrammatiken geführt und somit ein lohnendes Betätigungsfeld für die vorliegende Dissertation bereithalten.

In einer Arbeit von Brayer/Fu [BrFu 75] über Graphgrammatiken und ihre Anwendung für die Mustererkennung finden sich zwar einige Gedanken über prinzipielle Möglichkeiten, das Problem

der Inferenz von Graphgrammatiken durch Bereitstellung von speziellem a-priori-Wissen über den Aufbau der Muster oder unter Zuhilfenahme eines menschlichen Lehrers auf bereits früher realisierte Konzepte zurückzuführen, doch bei näherer Betrachtung erweisen sich all diese Vorschläge als höchst problematisch im Hinblick auf die Erfüllbarkeit ihrer Voraussetzungen bzw. ihrer zu erwartenden Rechenzeitanforderungen.

Der erste Vorschlag von Brayer/Fu läuft beispielsweise darauf hinaus, eine vorhandene Stichprobe von Graphen in Bäume umzuwandeln und dann das Baumgrammatik-Inferenzverfahren von Bhargava/Brayer/Fu anzuwenden. Für die Umwandlung wird empfohlen, den Aufbau eines Graphen aus a-priori bekannten Teilstrukturen verschiedener Komplexität ähnlich wie bei einem Ableitungsbaum darzustellen. Hierzu wäre anzumerken, daß die Extraktion des geforderten a-priori-Wissens aus der Stichprobe und die Umwandlung aller Graphen in Bäume soviel Arbeit machen wird, daß man hier nicht mehr von einem automatischen Verfahren sprechen kann. Außerdem funktioniert die Umwandlung von Graphen in Bäume nur für Muster, die in schwach zusammenhängende Teilmuster zerlegbar sind und bei denen die Zusammensetzung aus Teilstrukturen keiner weiteren Beschreibung bedarf, d.h. für Muster, bei denen eine Darstellung in Form von Bäumen von vornherein möglich und sinnvoll gewesen wäre.

Der zweite Vorschlag orientiert sich an Winstons Konzept [Wi 70] zum Lernen von Strukturmodellen mit Hilfe eines sogenannten "semantischen Lehrers" und sieht vor, daß einem Programm zum Lernen einer Struktur ausgewählte Beispiele für die verschiedenen Erscheinungsformen der Struktur und sogenannte "near-misses", das sind Beispiele für ähnliche, aber nicht dazugehörige Strukturen, in einer sortierten Reihenfolge eingegeben werden und ein menschlicher Beobachter alle gelernten Unterstrukturen kontrolliert. Im Falle der Verwendung einer Graphgrammatik als Strukturmodell würde dies bedeuten, daß alle Teilstrukturen vor ihrer Umwandlung in ein nichtterminales Symbol geprüft und gegebenenfalls zurück-

gewiesen werden müßten. Dieser Ansatz versucht, Vorstellungen über den Lernvorgang beim Menschen auf eine Maschine zu übertragen, ist aber wegen der notwendigen Bereitstellung von near-misses und der ständigen Kontrolle des Lernvorgangs bestenfalls als halbautomatisches Verfahren zu bezeichnen.

Für den Fall, daß die ersten beiden Vorschläge sich wegen fehlender Voraussetzungen als nicht durchführbar erweisen sollten, schlagen Brayer/Fu den Einsatz von enumerativen Methoden möglichst unter Verwendung heuristischer Suchverfahren zur Aufwandsreduzierung und einer Kostenfunktion zur Ermittlung einer optimalen Grammatik vor. Es steht zwar außer Zweifel, daß so etwas mit dem entsprechenden Aufwand theoretisch realisierbar wäre, aber alle Erfahrungen mit enumerativen Zeichenkettengrammatiken lassen hierfür Rechenzeiten in der Größenordnung von Wochen oder Jahren erwarten und den letzten Vorschlag als derzeit absolut impraktikabel erscheinen.

Vor diesem Hintergrund wird verständlich, warum es der Entwicklung auf dem Gebiet der Inferenz von Grammatiken bis heute noch nicht gelungen ist, ein Niveau zu erreichen, das ihren routinemäßigen Einsatz für Anwendungsprobleme ermöglicht, obwohl das Interesse dafür durchaus vorhanden wäre. Die Hauptursache dafür ist wohl darin zu suchen, daß die Lücke zwischen dem, was ein Mustererkennungssystementwickler braucht, und dem, was ein Inferenzprogramm zu liefern imstande ist, noch zu groß ist, um eine Beschäftigung des ersteren mit dem letzteren zu motivieren. Gerade dies könnte aber nach Meinung führender Experten auf diesem Gebiet einen Aufschwung für die gesamte syntaktische Mustererkennung nach sich ziehen:

"It seems that the latter problem (grammatical inference, Anm.d.V.) is the major remaining obstacle in the widespread use of syntactic techniques".
[PaAl 79]

Die vorliegende Arbeit ist ein Versuch, die geschilderte Lücke zu verkleinern durch die Entwicklung eines neuen

Insatzes zur Strukturerkennung komplexer Linienmuster mit Hilfe von speziellen Graphgrammatiken. Die im Rahmen dieser Arbeit erstellten Computerprogramme ermöglichen die Gewinnung einer Graphgrammatik für Stichproben von gerichteten Graphen mit Knoten- und Kantenmarkierung und die Analyse weiterer Graphen auf der Basis dieser Graphgrammatik, ohne daß der Benutzer fundierte Kenntnisse über Graphersetzungssysteme [Na 79] besitzen muß. Ein zwar aufwendig zu berechnendes, aber einfach zu handhabendes Gütekriterium erlaubt die Beurteilung der Eignung einer Graphgrammatik für eine gegebene Aufgabenstellung, so daß ohne großen Aufwand mit dem System auch experimentiert werden kann.

Zum Abschluß der Einleitung folgt ein kurzer *Überblick über die Arbeit.*

Kapitel 2 dient der Einführung von *Bezeichnungen* und der Bereitstellung *theoretischer Grundlagen*, die für die folgenden Kapitel der Arbeit von Bedeutung sind. Die Gliederung in einen Unterabschnitt zum Bereich Graphen und Graphgrammatiken und einem Unterabschnitt zum Bereich Inferenz und Syntaxanalyse soll eine Aufteilung in Arbeitsmaterial und Hilfsmittel zu seiner Bearbeitung widerspiegeln.

Kapitel 3 beschäftigt sich mit dem zentralen Problem der *Inferenz von Graphgrammatiken*. Nach einem kurzen Exkurs über die Darstellung von Mustern in Form von gerichteten, knoten- und kantenmarkierten Graphen folgt eine Schilderung der Überlegungen und Erfahrungen, die zur gewählten Beschränkung der Produktionentypen und zur inkrementellen Vorgehensweise beim Inferenzalgorithmus geführt haben. Kapitel 3.2 beschreibt dann zwei realisierte Versionen eines konstruktiven Inferenzverfahrens für Graphgrammatiken, bringt Ergebnisbeispiele für beide Verfahren und versucht, aus den gewonnenen Erfahrungen Empfehlungen für weitere Anwendungen des Verfahrens abzuleiten.

Kapitel 4 behandelt die *Analyse von Graphen mit Hilfe einer Graphgrammatik*. Insbesondere wird ein genau analog zum Inferenzalgorithmus arbeitendes Syntaxanalyseverfahren beschrieben.

ben, das die Zugehörigkeit beliebiger Graphen zur Sprache einer vom Inferenzalgorithmus gelieferten Grammatik entscheiden und je nach Aufgabenstellung weitere Analyseergebnisse liefern kann. Die Anwendung des Verfahrens wird durch Beispiele illustriert.

Kapitel 5 ist der Erweiterung des Ansatzes auf *programmierte Graphgrammatiken* gewidmet und beschreibt nach einer kurzen Einführung des zusätzlichen Formalismus der Programmierung die Maßnahmen, die zur Inferenz von programmierten Graphgrammatiken und deren Einsatz für Analysezwecke zu treffen sind. Beispiele, Erfahrungen und ein Vergleich der Ergebnisse mit denen nichtprogrammierter Graphgrammatiken beschließen dieses Kapitel.

Kapitel 6 beschäftigt sich mit einer Erweiterung des Ansatzes auf *stochastische Graphgrammatiken* und stochastische Syntaxanalyse für programmierte Graphgrammatiken. Geschildert werden die Möglichkeiten der Schätzung der entsprechenden Wahrscheinlichkeiten aus der Stichprobe und deren Verwendung zur Steuerung des Analyseverfahrens. Beispiele, Erfahrungen und eine Bemerkung zu den Schwierigkeiten, ein tragfähiges Fundament für weitergehende statistische Aussagen zu schaffen, bilden den Abschluß dieses Kapitels.

Kapitel 7 behandelt das Problem der *Beurteilung der Güte einer Grammatik* in bezug auf die dem Inferenzverfahren zugrundeliegende Stichprobe. Hierzu werden drei einfache strukturelle Gütekriterien definiert, ihre Berechnung beschrieben und durch Ergebnisbeispiele illustriert. Außerdem wird auf die Problematik der Anwendung von statistischen Gütekriterien eingegangen.

Kapitel 8 bringt einen abschließenden *Vergleich* des Ansatzes und seiner Anwendungsmöglichkeiten *mit verwandten Arbeiten* und enthält ferner einige Anregungen für weitere Untersuchungen auf diesem Gebiet.

2. GRUNDLAGEN

Das folgende Kapitel gibt einen Überblick über die wichtigsten theoretischen Grundlagen dieser Arbeit und soll gleichzeitig der Einführung häufig verwendeter Begriffe und Bezeichnungen dienen.

Der erste Abschnitt 2.1 beschäftigt sich mit den Musterbeschreibungsfomalismen Graph und Graphgrammatik, der zweite Abschnitt 2.2 mit verschiedenen Möglichkeiten zur Informationsgewinnung aus und über Graphen und Graphgrammatiken durch Inferenz- und Syntaxanalyseverfahren.

2.1 GRUNDLAGEN ZUM BEREICH GRAPHEN UND GRAPHGRAMMATIKEN

2.1.1 GRAPHEN

Graphen sind ein wegen ihrer Anschaulichkeit und Präzision beliebtes und geeignetes Hilfsmittel zur Darstellung von Sachverhalten, die durch Objekte und Beziehungen zwischen diesen Objekten charakterisiert werden können.

Bezeichnet man mit

N eine endliche Menge von Objekten, die Knoten genannt wurden,

V_N ein Knotenmarkierungsalphabet und

V_E ein Kantenmarkierungsalphabet,

dann lassen sich die gebräuchlichsten Typen von Graphen wie folgt formal definieren:

Definition 1:

a) Ein *gerichteter, knoten- und kantenmarkierter Graph* X ist ein Quadrupel $X = (N, E, m_N, m_E)$ mit

N endliche, nichtleere Menge von Knoten

$E \subseteq N \times N$ Menge von geordneten Knotenpaaren, die Kanten genannt werden

$m_N: N \rightarrow V_N$ Knotenmarkierungsfunktion

$m_E: E \rightarrow V_E$ Kantenmarkierungsfunktion.

b) Bei einem gerichteten, nicht knoten- bzw. kantenmarkierten Graphen entfallen m_N bzw. m_E .

c) Bei einem ungerichteten Graphen ist E eine Menge von ungeordneten Knotenpaaren.

Im folgenden soll mit der Bezeichnung Graph immer ein gerichteter, knoten- und kantenmarkierter Graph gemeint sein, alle anderen Fälle werden durch entsprechende Attribute gekennzeichnet.

Die Elemente von N müssen durch verschiedene Bezeichnungen identifizierbar sein. Dies soll der Einfachheit halber o.B.d.A. durch eine injektive Knotenbezeichnungsfunktion der Form $b_N: N \rightarrow \mathbb{N}$ geschehen, d.h. die Bezeichnung einzelner Knoten erfolgt durch Numerierung.

Die graphische Darstellung von Graphen wird wie üblich so gestaltet, daß jeder Knoten durch einen Kreis repräsentiert wird, der innen mit seiner Markierung und außen mit seiner Bezeichnung beschriftet ist. Jede Kante wird durch einen Pfeil vom ersten zum zweiten Knoten des Paares dargestellt, der mit der entsprechenden Kantenmarkierung beschriftet ist.

Beispiel 1:

Der Graph $X = (N, E, m_N, m_E)$ mit

$$N = \{1, 2, 3, 4\}$$

$$E = \{(1,2), (2,3), (3,4), (4,1), (1,3), (2,4)\}$$

$$m_N(1) = A, m_N(2) = B, m_N(3) = C, m_N(4) = D$$

$$m_E(1,2) = \alpha, m_E(2,3) = \beta, m_E(3,4) = \gamma,$$

$$m_E(4,1) = \delta, m_E(1,3) = \epsilon, m_E(2,4) = \eta$$

ist übersichtlich darstellbar durch Bild 1

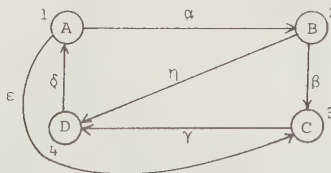


Bild 1

Darstellung eines Graphen

Ein Graph, der sich wie der aus Bild 1 in einer Ebene zeichnen läßt, ohne daß sich zwei Kanten überschneiden, heißt *planar*.

Die nun folgenden Definitionen zur Charakterisierung der Beziehungen zwischen zwei Graphen orientieren sich an [Na 79].

Definition 2:

Seien $X = (N, E, m_N, m_E)$, $X' = (N', E', m_N', m_E')$ und $X'' = (N'', E'', m_N'', m_E'')$ drei Graphen aus Menge $X(V_N, V_E)$ aller Graphen über dem Knotenmarkierungsalphabete V_N und dem Kantenmarkierungsalphabete V_E .

a) X' heißt *Teilgraph* von X (abgekürzt $X' \subseteq X$) gdw.

$$\begin{aligned} N' &\subseteq N \wedge E' \subseteq E \wedge m_N' = m_N|_{K'} \\ \wedge \forall (n_i, n_j) \in E' \quad m_N'(n_i, n_j) &= m_N(n_i, n_j). \end{aligned}$$

b) X' heißt *Untergraph* von X (abgek. $X' \subseteq X$) gdw.

$$\begin{aligned} N' &\subseteq N \wedge E' = E \cap (N' \times N') \wedge m_N' = m_N|_{K'} \\ \wedge \forall (n_i, n_j) \in E' \quad m_N'(n_i, n_j) &= m_N(n_i, n_j). \end{aligned}$$

c) X'' heißt zu X' *komplementärer Untergraph* in X (abgek.

$$X'' = X - X') \text{ gdw.}$$

$$X' \subseteq X \wedge X'' \subseteq X \wedge N'' = X \setminus X'.$$

Teilgraphen und Untergraphen entstehen durch Entfernen von Knoten und Kanten aus einem Graphen. Der Unterschied zwischen beiden besteht darin, daß bei einem Untergraphen nur Kanten mit gleichzeitig eliminiertem Anfangs- oder Endknoten, bei einem Teilgraphen auch Kanten mit verbleibenden Anfangs- und Endknoten entfernt werden dürfen.

Definition 3:

Seien X und X' zwei Graphen wie in Definition 2 spezifiziert.

a) X und X' heißen *isomorph* gdw.

- X und X' gleiche Knoten- und Kantenzahl besitzen und
 - eine bijektive Abbildung $f: N \rightarrow N'$ existiert, so daß
- $$\begin{aligned} \forall (n_i, n_j) \in E \exists (n_i', n_j') \in E' \text{ mit } f(n_i) &= n_i', f(n_j) = n_j' \\ \wedge m_E(n_i, n_j) &= m_E'(n_i', n_j'). \end{aligned}$$

b) X und X' heißen *äquivalent* gdw.

- X und X' gleiche Knoten- und Kantenzahl besitzen und
- eine bijektive Abbildung $f: N \rightarrow N'$ existiert, so daß

$$\begin{aligned} \forall n_i \in N \exists n_i' \in N' \text{ mit } f(n_i) = n_i' \wedge m_N(n_i) = m_N'(n_i') \text{ und} \\ \forall (n_i, n_j) \in E \exists (n_i', n_j') \in E' \text{ mit } f(n_i) = n_i', f(n_j) = n_j' \\ \wedge m_E(n_i, n_j) = m_E'(n_i', n_j'). \end{aligned}$$

Bei isomorphen Graphen können die Knoten unterschiedlich bezeichnet und markiert, bei äquivalenten Graphen nur unterschiedlich bezeichnet sein. Die Kantenstruktur muß in beiden Fällen völlig übereinstimmen.

Definition 4 dient zur Einführung von Begriffen für Eigenschaften von Knoten und Kanten eines Graphen nach [Ti 76].

Definition 4:

- a) Die Anzahl der Kanten eines Graphen mit Anfangsknoten n_i heißt Ausgangsgrad des Knotens n_i und wird mit $\gamma^+(n_i)$ bezeichnet.
- b) Die Anzahl der Kanten eines Graphen mit Endknoten n_i heißt Eingangsgrad des Knotens n_i und wird mit $\gamma^-(n_i)$ bezeichnet.
- c) Die Anzahl der Kanten eines Graphen mit Anfangs- oder Endknoten n_i heißt *Grad* des Knotens n_i und wird mit $\gamma(n_i)$ bezeichnet.
- d) Eine Kante heißt Schlinge, wenn Anfangs- und Endknoten der Kante identisch sind.
- e) Zwei Kanten heißen parallel, wenn sie die gleichen Anfangs- und die gleichen Endknoten besitzen.

Im folgenden sollen nur noch schlichte Graphen, d.h. Graphen ohne Schlingen und parallele Kanten betrachtet werden.

Definition 5 beschäftigt sich mit Verbindungen zwischen zwei Knoten.

Definition 5:

Sei $(n_1, n_2, \dots, n_k)$ eine Folge von Knoten in einem Graphen $X = (N, E, m_N, m_E)$ mit der Eigenschaft, daß zwischen je zwei aufeinanderfolgenden Knoten eine Kante existiert. Der durch die Knotenfolge und die beschriebenen Verbindungskanten erzeugte Teilgraph X_T von X heißt

- a) *ungerichteter Kantenzug* zwischen n_1 und n_k , wenn
 $\forall i \in \{1, 2, \dots, k-1\} \quad (n_i, n_{i+1}) \in E \vee (n_{i+1}, n_i) \in E$ gilt.
- b) *gerichteter Kantenzug* von n_1 nach n_k , wenn
 $\forall i \in \{1, 2, \dots, k-1\} \quad (n_i, n_{i+1}) \in E$ gilt.
- c) Ein ungerichteter Kantenzug heit *Weg*, wenn er keinen Knoten mehrfach enthlt.
- d) Ein gerichteter Kantenzug heit *Bahn*, wenn er keinen Knoten mehrfach enthlt.
- e) Die *Lnge* eines Weges bzw. einer Bahn ist identisch mit der Anzahl der in X_T enthaltenen Kanten.

Definition 6:

Ein Graph $X = (N, E, m_N, m_E)$ heit *zusammenhngend*, wenn es zwischen je zwei Knoten n_i und n_j aus N einen Weg gibt.

Definition 7:

Ein zusammenhngender Teilgraph X_T von X , der alle Knoten von X enthlt und eine minimale Anzahl von Kanten besitzt, heit *Gerst* von X .

Definition 8:

Ein Gerst, das einen Knoten enthlt, von dem aus Bahnen zu allen brigen Knoten existieren, heit *gerichtetes Gerst*.

Es gelten folgende Zusammenhnge:

Satz 1:

- a) Jeder zusammenhngende Graph besitzt mindestens ein (nicht notwendigerweise gerichtetes) Gerst.
- b) Jedes Gerst mit k Knoten enthlt genau $k-1$ Kanten.

Die Beweise hierfr sind einfach und z.B. in [De 74] oder [Ti 76] nachlesbar.

2.1.2 GRAPHGRAMMATIKEN

Graphgrammatiken sind als zweidimensionale Verallgemeinerung der Chomsky'schen Zeichenkettengrammatiken seit ca. zehn Jahren

[PfRo 69] Gegenstand theoretischer Untersuchungen und praktischer Anwendungen. Die theoretischen Untersuchungen haben eine Reihe von verschiedenen Ansätzen zur Spezifikation der sogenannten Einbettungsüberführung, d.h. der Vorschriften darüber, wann und wie eine Produktion zur Veränderung eines Graphen angewendet werden darf, hervorgebracht und darauf aufbauend eine der Theorie der formalen Zeichenkettensprachen vergleichbare Theorie der Graphsprachen entwickelt. Die praktischen Anwendungen dienen hauptsächlich zur Modellierung von statischen oder dynamisch sich verändernden Strukturen. Einen umfassenden Überblick über das gesammelte Wissen zum Thema Graph-Ersetzungssysteme gibt das Buch [Na 79], an dem sich auch die folgenden Definitionen orientieren.

Definition 9:

Eine *Graphgrammatik* ist ein Tupel $G = (V_N, V_E, V_{NT}, V_{ET}, X_O, P)$ mit

- a) $V_{NT} \subseteq V_N$ und $V_{ET} \subseteq V_E$ heißen terminales Knoten - bzw. Kantenmarkierungsalphabet, $V_N \setminus V_{NT}$ und $V_E \setminus V_{ET}$ nichtterminales Knoten- bzw. Kantenmarkierungsalphabet.
- b) $X_O \in X(V_N, V_E)$ heißt Startgraph und muß mindestens einen nicht-terminalen Knoten enthalten.
- c) P ist eine endliche Menge von Produktionen der Form $p = (X_1, X_r, E_g)$ mit X_1 als zu ersetzendem Graphen, der wieder mindestens einen nichtterminalen Knoten enthält, X_r als dafür einzusetzendem Graphen und E_g als Einbettungsvorschrift.

Nach [Na 79] kann für beliebige Graphgrammatiken o.B.d.A. angenommen werden, daß der Startgraph X_O nur aus einem nicht-terminalen Knoten besteht. Im folgenden sollen deshalb nur noch Grammatiken mit einknotigem Startgraphen betrachtet werden.

Die Alphabete V_{NT} , V_{ET} und $V_N \setminus V_{NT}$ dürfen nicht leer sein, wenn mit der Graphgrammatik terminale Graphen, d.h. Graphen aus der Menge $X(V_{NT}, V_{ET})$ erzeugbar sein sollen.

Der Begriff der Ableitbarkeit kann ohne konkrete Angabe einer

Einbettungsvorschrift nur unvollständig definiert werden. Nachdem aber die existierenden Ansätze für die Einbettungsvorschriften außerordentlich vielfältig, komplex und bisher nicht mit einem gemeinsamen Formalismus darstellbar sind, soll an dieser Stelle nur festgelegt werden, daß eine Einbettungsvorschrift E_g sowohl Bedingungen über die Anwendbarkeit einer Produktion als auch Anweisungen für deren Ausführung enthalten darf, die über eine bloße Ersetzung von X_1 durch X_r hinausgehen können.

Definition 10:

Ein Graph X_{i+1} heißt aus dem Graphen X_i mittels der Produktion $p = (X_1, X_r, E_g)$ *direkt ableitbar* (abgek. $X_i \xrightarrow{p} X_{i+1}$) gdw.

- $X_1 \subseteq X_i$
- $X_r \subseteq X_{i+1}$ und eventuelle weitere Anweisungen für die Generierung von Verbindungskanten zwischen X_r und $X_{i+1} - X_r$ ausgeführt sind (siehe dazu z.B. [Na 79])
- $X_i - X_1 = X_{i+1} - X_r$.

Die aus dem Startgraphen einer Graphgrammatik mittels irgendeiner Folge von Produktionen ableitbaren Graphen heißen Graph-Satzformen.

Die terminalen unter ihnen bilden die *Sprache der Graphgrammatik*

$$L(G) = \{ X \mid X \in X(V_{NT}, V_{ET}) \wedge X_0 \Rightarrow X \}.$$

Zwei Graphgrammatiken G_1 und G_2 heißen äquivalent gdw.

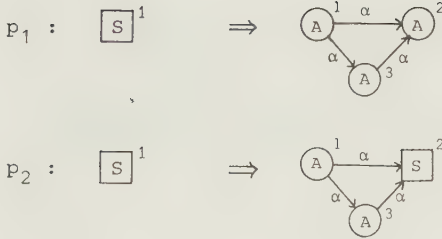
$L(G_1) = L(G_2)$, d.h. wenn sie dieselbe Sprache erzeugen.

Zur Veranschaulichung soll das folgende Beispiel für eine Graphgrammatik dienen, wobei nichtterminale Knoten als rechteckige Kästchen und terminale Knoten als Kreise dargestellt werden.

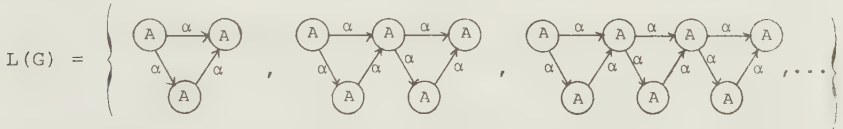
Beispiel 2:

$G = (\{A, S\}, \{a\}, \{A\}, \{a\}, X_0, \{p_1, p_2\})$ mit

$X_0 = \boxed{S}$ und



Für die Einbettungsvorschrift E_g soll gelten, daß die Anwendbarkeit einer Produktion nur vom Erfülltsein der Bedingung $X_l \subseteq X_1$ abhängt und alle in einen nichtterminalen Knoten hineinlaufenden Kanten auch bei dessen Umwandlung in einen terminalen Knoten erhalten bleiben. Die Sprache der obigen Grammatik ist dann:



Durch Einschränkungen für die Gestalt der linken und rechten Seite der Produktionen einer Graphgrammatik läßt sich analog zum Zeichenkettenfall eine Hierarchie von Graphsprachen aufbauen.

Definition 11:

Sei $p = (X_l, X_r, E_g)$ eine Graphproduktion.

- p heißt *monoton* gdw. die Anzahl der Knoten in X_r nicht kleiner ist als die Anzahl der Knoten in X_l .
- p heißt *kontextsensitiv* gdw. ein einknotiger nichtterminaler Graph $X_l' \subseteq X_l$ und ein nichtleerer Graph $X_r' \subseteq X_r$ existieren, für die $X_l - X_l' = X_r - X_r'$ gilt und außerdem E_g den Kontext $X_l - X_l'$ unverändert läßt.
- p heißt *kontextfrei* gdw. p kontextsensitiv ist und X_l nur einen Knoten enthält.

- d) p heißt *linear* gdw. p kontextfrei ist und X_r höchstens einen nichtterminalen Knoten enthält.
- e) p heißt *regulär* gdw. p linear ist und der eventuell existierende nichtterminale Knoten keine auslaufenden Kanten besitzt.

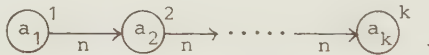
Eine Graphgrammatik heißt uneingeschränkt, monoton, kontextsensitiv, kontextfrei, linear oder regulär gdw. alle ihre Produktionen diese Eigenschaft besitzen.

Die Graphgrammatik aus Beispiel 2 ist demzufolge regulär.

Die Einschränkungen der Graphproduktionen orientieren sich völlig an den entsprechenden Einschränkungen im Zeichenkettenfall.

Satz 2:

Jede Zeichenkette $a_1 a_2 \dots a_k$ ist in Graphform darstellbar als *Wortpfad*



Es gibt nun zu jeder Zeichenkettengrammatik G vom Typ uneingeschränkt, monoton, kontextsensitiv, kontextfrei (ohne Löschung), linear oder regulär eine Graphgrammatik G' , die genau die Wortpfade von $L(G)$ erzeugt und in der Hierarchie der Graphgrammatiken demselben Typ angehört.

Ein formaler Beweis dieses intuitiv einsichtigen Zusammenhangs ist in [Na 79] nachzulesen.

Bezeichnet man mit UG , MG , KSG , KFG , LG und RG die Klasse der uneingeschränkten, monotonen, kontextsensitiven, kontextfreien, linearen und regulären Graphsprachen, so gilt nach Definition 11 folgende Beziehung:

$$UG \supseteq MG \supseteq KSG \supseteq KFG \supseteq LG \supseteq RG$$

Die Echtheit der Inklusion $MG \subset UG$ ist durch die Klasse der aufzählbaren Zeichenketten Sprachen, deren äquivalente Graphgrammatiken nicht in MG enthalten sind, leicht nachweisbar. Für die übrigen Inklusionsbeziehungen sind die Erfahrungen aus

dem Zeichenkettenfall weniger leicht übertragbar, weil nicht zuletzt die Mächtigkeit der Einbettungsüberführung das Ergebnis entscheidend mitbeeinflussen kann. Dies soll jedoch an dieser Stelle nicht weiter erörtert werden.

Für die weitere Arbeit von Bedeutung sind noch die folgenden Ergebnisse:

Satz 3:

- a) Jede Graphsprache ist aufzählbar.
- b) Jede monotone Graphsprache ist entscheidbar.

Die Beweise hierfür verlaufen vollkommen analog zum Zeichenkettenfall und können deshalb unterbleiben.

Satz 4:

Für monotone Graphgrammatiken G , G_1 und G_2 ist i.a. nicht entscheidbar, ob

- a) $L(G)$ endlich ist
- b) $L(G_1) = L(G_2)$ ist .

Dies sind direkte Folgerungen aus der Unentscheidbarkeit der entsprechenden Probleme im Zeichenkettenfall.

2.2 GRUNDLAGEN ZUM BEREICH INFERENZ VON GRAMMATIKEN UND SYNTAXANALYSE

Zur Verdeutlichung der Stellung von Inferenz- und Syntaxanalyseverfahren in einem syntaktischen Mustererkennungssystem soll das folgende Blockdiagramm beitragen:

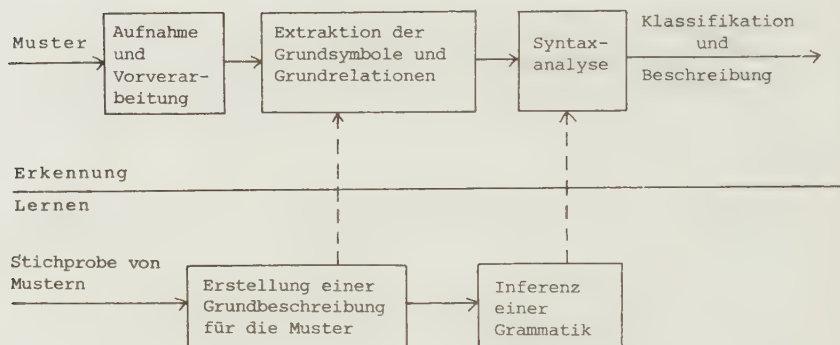


Bild 2

Struktur eines syntaktischen Mustererkennungssystems

Sowohl Inferenz- als auch Syntaxanalyseverfahren dienen zur Gewinnung von struktureller Information über die Muster eines bestimmten Problemkreises. Inferenzverfahren werden in der Entwicklungsphase eines Mustererkennungssystems eingesetzt, um eine Grammatik als Strukturmodell für eine bekannte Menge von Mustern zu gewinnen; Syntaxanalyseverfahren werden im späteren Mustererkennungssystem für die Feststellung verwendet, ob irgendein zu verarbeitendes Muster mit der inferierten Grammatik erzeugbar und damit dem bekannten Problemkreis von Mustern zugehörig ist.

2.2.1 INFERENZ VON GRAMMATIKEN

Unter *Inferenz von Grammatiken* (im folgenden meist nur Inferenz

genannt) versteht man die Aufgabe, zu einer gegebenen Stichprobe von Sätzen einer (ein- oder mehrdimensionalen) formalen Sprache eine Grammatik zu konstruieren, die alle Sätze der Stichprobe und weitere ähnlich strukturierte Sätze erzeugen kann.

Zur Untersuchung der Frage, ob und wie weit dies überhaupt möglich ist, bedarf es einer stärkeren mathematischen Formalisierung des Problems.

Definition 12:

- a) Eine Menge von Sätzen S^+ heißt *positive Stichprobe* einer formalen Sprache $L(G)$, wenn $S^+ \subseteq L(G)$.
- b) Eine Menge von Sätzen S^- heißt *negative Stichprobe* von $L(G)$, wenn $S^- \subseteq \overline{L(G)}$.
- c) S^+ heißt *vollständig*, wenn $S^+ = L(G)$.
- d) S^- heißt *vollständig*, wenn $S^- = \overline{L(G)}$.
- e) S^+ heißt *strukturell vollständig*, wenn jede Produktion von G zur Erzeugung von mindestens einem Satz aus $L(G)$ verwendet wird.

Mit diesen Definitionen läßt sich das Inferenz-Problem weiter präzisieren.

Hierzu die folgende Skizze:

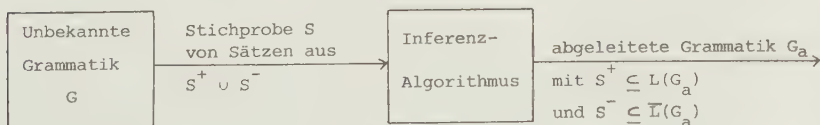


Bild 3

Modell eines Inferenzsystems

Man geht von der Vorstellung aus, daß eine unbekannte Grammatik G eine Sprache $L(G)$ erzeugt, von der eine Stichprobe von Sätzen $x_i \in L(G)$ und eine (evtl. leere) Stichprobe von Sätzen $y_i \in \overline{L(G)}$

bekannt sind. Die gestellte Aufgabe besteht nun darin, aus der gegebenen Stichprobe eine Grammatik G_a abzuleiten, die ein möglichst gutes Modell für die Stichprobe darstellt und für die mindestens gilt:

$$S^+ \subseteq L(G_a) \quad \text{und} \quad S^- \subseteq \overline{L}(G_a) \quad .$$

Erfüllt eine Grammatik diese beiden Bedingungen, so heißt sie mit der Stichprobe *kompatibel*.

Um die Kompatibilität einer Grammatik mit einer Stichprobe überprüfen zu können, ist es unbedingt notwendig, daß in der Sprache $L(G_a)$ das *Satzproblem entscheidbar* ist. Nachdem dies für alle monotonen Zeichenketten- und Graphsprachen (siehe Satz 3) immer gewährleistet ist, stellt die aus dieser Forderung resultierende Beschränkung auf Sprachen, in denen das Satzproblem entscheidbar ist, keinen spürbaren Nachteil für die Auswahl von Grammatiken dar.

Das höchste Ziel für einen Inferenz-Algorithmus wäre die exakte Rekonstruktion der unbekannten Grammatik G aus der Stichprobe. Nachdem es aber keine eindeutige Beziehung zwischen einer formalen Sprache und einer Grammatik gibt, sondern jede Sprache von vielen verschiedenen Grammatiken erzeugt werden kann, ist dieses Ziel nicht erreichbar.

Das nächstniedrigere Ziel wäre, eine zu G äquivalente Grammatik G_a zu konstruieren. Die Voraussetzungen dafür spezifiziert der folgende Satz:

Satz 5: (nach [Ma 74])

- a) Für eine endliche Sprache $L(G)$ ist es mit Hilfe einer vollständigen, positiven Stichprobe S^+ möglich, eine Grammatik zu konstruieren, für die $L(G_a) = L(G)$ gilt.
- b) Für eine unendliche Sprache $L(G)$ ist es mit Hilfe einer endlichen Stichprobe S^+ oder einer endlichen, klassifizierten Stichprobe $S^+ \cup S^-$ i.a. nicht möglich, eine Grammatik mit der Eigenschaft $L(G) = L(G_a)$ abzuleiten.

Zum Beweis:

- a) Sei $S^+ = L(G) = \{x_i \mid i \in 1, 2, \dots, k\}$ eine Menge von Zeichen-

ketten über einem Alphabet V_T bzw. eine Menge von Graphen über den Knoten- und Kantenmarkierungsalphabeten V_{NT} und V_{ET}^* . Wegen der Endlichkeit der Menge $L(G)$ ist eine triviale Grammatik G_a mit dem Startsymbol S_a und der Produktionsmenge $P_a = \{S_a \Rightarrow x_i \mid i = 1, 2, \dots, k\}$ leicht aufstellbar und es gilt $L(G_a) = L(G)$.

- b) Notwendige Voraussetzung für die Konstruktion einer zu G äquivalenten Grammatik G_a wäre die Berechenbarkeit einer Entscheidungsfunktion der Form

$$f(z_i) = \begin{cases} +1 & \text{für } z_i \in S^+ \\ -1 & \text{für } z_i \in S^- \end{cases}$$

für alle Zeichenketten $z_i \in V_T^*$ bzw. alle Graphen $z_i \in X(V_{NT}, V_{ET})$ [Lo 72]. Wegen der Unvollständigkeit der Stichprobe gibt es aber $z_i \in V_T^* \setminus (S^+ \cup S^-)$ bzw. $X(V_{NT}, V_{ET}) \setminus (S^+ \cup S^-)$, deren Zugehörigkeit zu S^+ oder S^- weder bekannt noch berechenbar ist und die die obige Voraussetzung unerfüllbar werden lassen.

Es ist an dieser Stelle erwähnenswert, daß diese Ergebnisse allein auf der Eigenschaft von S^+ und S^- basieren, Untermengen von V_T^* bzw. $X(V_{NT}, V_{ET})$ zu sein und von der speziellen Form der Grammatik G_a unabhängig sind.

Die beiden Aussagen des letzten Satzes geben Aufschluß über die theoretische Lösbarkeit bzw. Unlösbarkeit des Inferenz-Problems in Abhängigkeit von der Vollständigkeit bzw. Unvollständigkeit der Stichprobe.

Für praktische Probleme sind die Voraussetzungen von Satz 5 a) und b) jedoch eher untypisch, weil eine vollständige positive Stichprobe wie in Satz 5 a) nur in wenigen Ausnahmefällen bekannt sein dürfte und es dann höchst ungeschickt wäre, eine Grammatik wie die im Beweis von Satz 5 a) angegebene zu konstruieren. Die Voraussetzung einer unendlichen Sprache $L(G)$ wie in Satz 5 b) ist auch praxisfern, weil es durch die maximale Größe der verarbeitbaren Muster immer mög-

lich sein wird, die Menge $L(G)$ als endlich anzunehmen. Aber unabhängig von der Endlichkeit oder Unendlichkeit von $L(G)$ bleibt das Hauptproblem die normalerweise unvollständige Stichprobe, die die Inferenz einer zu G äquivalenten Grammatik zu einem in der Regel nicht erreichbaren und nicht verifizierbaren Ziel macht. Interessant ist auch die Feststellung, daß die zusätzliche Kenntnis einer unvollständigen negativen Stichprobe das Inferenz-Problem nicht lösbarer macht, sondern lediglich zur Aussortierung von Grammatiken, die neben S^+ auch Elemente von S^- erzeugen, benutzt werden kann.

So bleibt bei praktischen Problemen im allgemeinen nichts anderes zu tun übrig als zu prüfen, ob eine abgeleitete Grammatik G_a mit der vorhandenen unvollständigen Stichprobe kompatibel, d.h. ob $S^+ \subseteq L(G_a)$ und $S^- \subseteq \overline{L}(G_a)$ ist.

Ein Versuch, wenigstens die Forderung nach einer für einen Problembereich von Mustern einigermaßen repräsentativen Stichprobe zu unterstreichen, ist die Forderung nach einer strukturell vollständigen Stichprobe (siehe Definition 12 e). Diese Eigenschaft ist zwar bei unbekannter Grammatik G nicht nachprüfbar, stellt aber eine Art Minimalforderung für eine sinnvolle Stichprobe dar, die man bei der Sammlung bzw. Auswahl von Stichprobenelementen soweit wie möglich zu berücksichtigen versuchen sollte.

Abschließend sei bemerkt, daß es aufgrund der i.a. nur näherungsweisen Lösbarkeit des Inferenz-Problems zu jeder Stichprobe S eine Menge nichtäquivalenter, mit S kompatibler Grammatiken gegeben wird, die es erforderlich machen können, nach im Hinblick auf den Verwendungszweck der Grammatik definierten Kriterien eine bestgeeignete Grammatik auszuwählen.

2.2.2 SYNTAXANALYSE

Das Ziel bei der Konstruktion einer Grammatik für eine Menge von Mustern ist es, diese Grammatik später für die Erkennung weiterer zur Sprache der Grammatik gehörender Muster verwenden

zu können. Dies kann bei endlichen Sprachen zwar auch durch Vergleich einer Musterbeschreibung x mit schlimmstenfalls allen Elementen einer Sprache $L(G_1)$ geschehen; diese Vorgehensweise ist aber für unendliche Sprachen nicht mehr anwendbar und außerdem höchst uneffizient. Deshalb versucht man, die kompaktere Beschreibung der Sprache in Form der Grammatik G_1 in einen Algorithmus umzusetzen, der entscheiden kann, ob eine Musterbeschreibung x durch eine Grammatik G_1 erzeugt werden kann oder nicht. Ein Algorithmus, der dies leistet, wird als Rekognitions- [Sc 75] oder *Syntaxanalyseverfahren* bezeichnet.

Bei Vorliegen eines Mehrklassenproblems kann man für jede Musterklasse eine eigene Grammatik G_i konstruieren und eine Musterbeschreibung x auf Erzeugbarkeit durch eine ganze Reihe von Grammatiken testen. Zur Veranschaulichung dieses Vorgangs soll das folgende Blockdiagramm dienen:

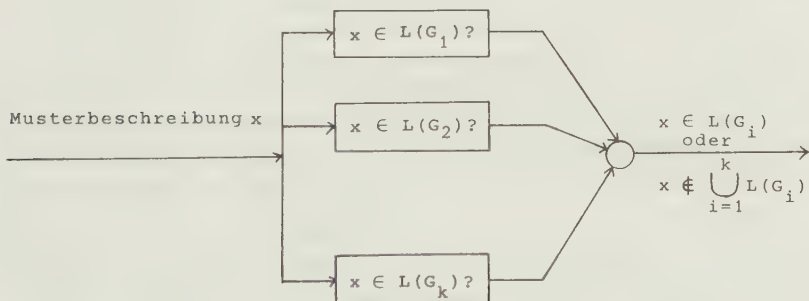


Bild 4

Struktur eines syntaktischen Klassifikationssystems

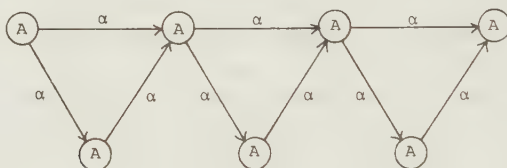
Die Klassifikation eines Musters auf syntaktischem Wege ist zurückführbar auf die Beantwortung der Frage, ob eine Musterbeschreibung x durch eine der Grammatiken $G_1, G_2, \dots, G_k$ erzeugbar ist und falls ja, durch welche. Dieses Problem ist genau dann eindeutig lösbar, wenn jedes der Teilprobleme

$x \in L(G_i)$ entscheidbar ist und die verschiedenen Sprachen $L(G_i)$ untereinander disjunkt sind, d.h. wenn $L(G_i) \cap L(G_j) = \emptyset$ für alle $i \neq j$ gilt.

Die Lösung eines Teilproblems der Form $x \in L(G_i)$ braucht nicht allein in einer Ja/Nein-Entscheidung zu bestehen, sondern kann für den Fall $x \in L(G_i)$ auch eine *Rekonstruktion des Ableitungsweges* liefern. Aus der Kenntnis des Ableitungsweges lassen sich weitere Informationen über in dem Muster x enthaltene Teilstrukturen gewinnen, die je nach Anwendungsfall z.B. im n-maligen Vorkommen von Produktion p_i oder der Produktionsfolge $p_i p_j p_k$ bestehen können.

Beispiel 3:

Das Muster

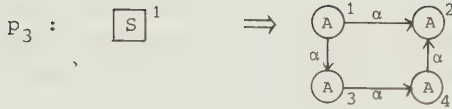


gehört zur Sprache $L(G)$ der Grammatik G aus Beispiel 2. Es ist erzeugbar durch die Produktionsfolge $p_2 p_2 p_1$, woraus in diesem speziellen Fall z.B. die Anzahl der im Muster enthaltenen Dreiecke direkt ablesbar ist.

Falls sich die Ableitungswege verschiedener Klassen von Mustern eindeutig voneinander unterscheiden lassen, kann es unter Umständen günstiger sein, die verschiedenen Grammatiken G_i zu einer einzigen Grammatik G mit $L(G) = \bigcup_{i=1}^k L(G_i)$ zusammenzufassen und durch diese Grammatik G nur die nicht klassifizierbaren Muster aussortieren bzw. die Produktionsfolge zur Erzeugung eines Musters erstellen zu lassen, aus der dann in einem zweiten Schritt die Klassenzugehörigkeit ermittelt werden muß.

Beispiel 4:

Erweitert man die Grammatik aus Beispiel 2 um die Produktion



so lassen sich die Menge der aus einer Kette von Dreiecken bestehenden Graphen und die Menge der aneinandergereihten Dreiecke mit einem Viereck am Ende wohl am günstigsten durch eine Graphgrammatik mit der Produktionsmenge $\{p_1, p_2, p_3\}$ auf syntaktische Richtigkeit überprüfen und anhand der letzten Produktion der Ableitungswege $p_2^n p_1$ bzw. $p_2^n p_3$ voneinander unterscheiden.

Die Beispiele 3 und 4 können natürlich nur eine vage Vorstellung von den Möglichkeiten vermitteln, die man im konkreten Einzelfall haben kann, um aus der Rekonstruktion des Ableitungsweges einer Musterbeschreibung die für die weitere Verarbeitung wesentliche Information zu gewinnen. Nachdem weitere Einzelheiten hierzu aber nur im Zusammenhang mit einer konkreten Aufgabenstellung sinnvoll erörterbar sind, soll dieser Abschnitt über Einsatzmöglichkeiten der Syntaxanalyse für die syntaktische Mustererkennung abgeschlossen werden mit der Bemerkung, daß man die Syntaxanalyse ganz allgemein als Komprimierung der in einer evtl. mehrdimensionalen Musterbeschreibung enthaltenen Information auf eine Ja/Nein-Entscheidung und eine Zeichenkette von Produktionsnummern des Ableitungsweges betrachten kann.

Die theoretische Entscheidbarkeit des Problems $x \in L(G_1)$ sagt zunächst noch nichts darüber aus, wie man die Zugehörigkeit von x zu $L(G_1)$ konkret nachprüfen kann. Hierfür gibt es abhängig von der Klasse der Grammatiken G_1 eine Reihe von unterschiedlich leistungsfähigen erkennenden Automaten, die so konstruiert werden können, daß sie genau die von einer gegebenen Grammatik G_1 erzeugte Sprache akzeptieren.

Ein erkennender Automat besteht in der Regel aus einem Eingabeband, einer endlichen Menge von Zuständen und einer Spei-

herungsmöglichkeit. Die einzelnen Stationen des Erkennungsprozesses sind charakterisierbar durch Konfigurationen, aus denen der Zustand des Automaten, der Inhalt des Eingabebandes und des Speichers hervorgehen. Die Erkennung eines Satzes x einer formalen Sprache $L(G_1)$ beginnt mit einer definierten Anfangskonfiguration und endet durch Erreichen einer Endkonfiguration, aus der die Entscheidung, ob $x \in L(G_1)$ ist, ablesbar sein muß. Die Konfigurationen eines Automaten kann man als analog zu den Satzformen einer Grammatik während des Ableitungsvorgangs ansehen. Die Übergänge zwischen zwei Konfigurationen werden beschrieben durch eine den Produktionen einer Grammatik entsprechende Übergangsfunktion, die in Abhängigkeit vom Zustand des Automaten, dem Inhalt des gelesenen Eingabebandes und des Speichers bestimmte Aktionen zur Veränderung der Konfiguration durchführt. Solche Aktionen können z.B. in einem Weiterrücken des Lesekopfes auf dem Eingabeband, einer Veränderung des Zustands und/oder des Speicherinhalts bestehen. Der Übergang von einer Konfiguration zu einer anderen heißt *deterministisch*, wenn es genau eine Nachfolgekonfiguration gibt und *nichtdeterministisch*, wenn es verschiedene mögliche Nachfolgekonfigurationen gibt. Der Automat selbst heißt deterministisch, wenn alle definierten Übergänge zwischen zwei Konfigurationen deterministisch erfolgen.

Bei deterministischen Automaten verläuft die Erkennung eines Satzes einer formalen Sprache *sackgassenfrei*, bei nichtdeterministischen Automaten kann es vorkommen, daß an einer Stelle im Erkennungsprozeß eine zu diesem Zeitpunkt noch nicht erkennbare "falsche" Nachfolgekonfiguration gewählt wird und das Verfahren in eine Sackgasse gerät. Deswegen muß bei nichtdeterministischer Vorgehensweise soviel Information über die Vorgängerkonfigurationen gespeichert werden, wie man benötigt, um den Weg in die Sackgasse rückgängig zu machen (*back-up*) und einen anderen Weg einzuschlagen.

Die bekanntesten Beispiele für erkennende Automaten sind endliche Automaten ohne Speichermöglichkeit zur Erkennung regulärer Zeichenkettensprachen und Kellerautomaten mit Kellerspeicher, die kontextfreie Zeichenkettensprachen akzeptieren [HoU1 69].

Bereits die Erkennung von uneingeschränkt kontextfreien Zeichenkettensprachen ist i.a. nicht mehr deterministisch und damit sackgassenfrei möglich. Im Bereich der zweidimensionalen formalen Sprachen gibt es Baumautomaten für Baumsprachen [Do 70] und Graphautomaten für Graphsprachen von stark zusammenhängenden, knoten- und kantenmarkierten gerichteten Graphen [My 72] oder knoten- und kantenmarkierten ungerichteten Graphen [Mi 75]. Die beiden zuletzt genannten Automaten arbeiten ähnlich wie Turingmaschinen mit begrenztem Band, nur mit dem Unterschied, daß Graphen statt Zeichenketten abzusuchen und zu verändern sind, wodurch sich der Erkennungsprozeß erheblich langwieriger und komplizierter gestaltet.

Die Übergangsfunktion eines Automaten stellt eine Art Programm für sein Verhalten dar und kann als Grundlage für ein die gleichen Erkennungsfähigkeiten besitzendes Syntaxanalyseverfahren verwendet werden. Zusätzlich kommen aus Effizienzgründen meist noch strategische Überlegungen ins Spiel. Die Erkennung eines Satzes einer Sprache kann im wesentlichen in zwei Richtungen verlaufen. Die eine Richtung versucht, aus dem Startsymbol der Grammatik durch Anwendung von Produktionen den Satz x aufzubauen (*top-down-Strategie*), die andere Richtung will den Satz x durch umgekehrte Anwendung der Produktionen, wobei die vorhandene rechte Seite durch die linke ersetzt wird, auf das Startsymbol reduzieren (*bottom-up-Strategie*). Auch Mischformen aus beiden Strategien sind verwendbar [Sc 75].

Weitere Möglichkeiten, die Syntaxanalyse vor allem bei nicht-deterministisch arbeitenden Verfahren zu beschleunigen, bieten die Überprüfung der Produktionen in einer möglichst schnell zum Ziel führenden Reihenfolge (siehe Kapitel 4), die Verwendung einer programmierten Grammatik (Kapitel 5) oder stochastische Syntaxanalyse (Kapitel 6).

3. INFERENZ VON GRAPHGRAMMATIKEN

Dieses Kapitel beschreibt die Entwicklung von zwei Inferenzverfahren für spezielle Graphgrammatiken. Der Abschnitt 3.1 enthält Vorbemerkungen zur Stichprobe, zur Form der Produktionen und zur Inferenzstrategie, der Abschnitt 3.2 beschreibt die implementierten Inferenzalgorithmen, ihre Anwendung auf verschiedene Beispielstichproben und die dabei gewonnenen Erfahrungen.

3.1 ALLGEMEINE VORBEMERKUNGEN

Als erste Vorbemerkung soll ein nach wie vor gültiges Zitat aus [BrFu 75] angeführt werden:

"One of the most difficult and most intriguing problems associated with the linguistic approach to pattern recognition is that of constructing a grammar which adequately describes the patterns being considered. Many researchers have tackled the problem of trying to find automated schemes for finding a grammar from a subset of a language. Considering the difficulty that researchers themselves have in finding good grammars for non-trivial problems, it is not too surprising that these efforts have been of little practical value. Nevertheless these attempts are important because they yield insight into the detailed mechanisms of learning, and because they may eventually lead to successful results".

Diese Zeilen sollen verdeutlichen, daß die Entwicklung eines Inferenzverfahrens für Graphgrammatiken auf der Basis der wenig positiven Erfahrungen mit der Inferenz von anderen Grammatiktypen [BiFe 72, Hu 75, FuBo 75, Ba 78] und des teilweise noch etwas lückenhaften Unterbaus einer Theorie der Graphsprachen [Na 79] als durchaus ambitioniertes Unterfangen anzusehen ist, bezüglich dessen allgemeiner und zufriedenstellender Lösbarkeit man zumindest in absehbarer Zeit keine allzu großen Erwartungen hegen sollte.

Daß es dennoch möglich ist, durch Einschränkung der Aufgabenstellung in einer die Verwendbarkeit des Ansatzes für die

grammatikalische Mustererkennung möglichst wenig beeinträchtigenden Weise zu praktikablen Lösungen zu kommen, soll diese Arbeit demonstrieren.

Die wichtigsten globalen Entwurfskriterien für ein Verfahren zur Inferenz von auf Zwecke der Bildverarbeitung geeignet einzuschränkenden Graphgrammatiken waren:

1. Anschaulichkeit

- a) Der Übergang von einem Bild zu einem es beschreibenden Graphen und zurück soll sowohl manuell als auch automatisch problemlos möglich sein.
- b) Einer inferierten Grammatik sollte mit wenig Übung anzusehen sein, wie der Erzeugungsprozeß der Graphen funktioniert, d.h. wann welche Produktion anwendbar ist und welche Veränderungen sie bewirkt.

Negativbeispiele hierfür wären zu a) die PDL-Beschreibungen mehrdimensionaler Muster oder zu b) z.B. die Grammatik zur Erzeugung aller Wälder von Bäumen in [Na 79] auf S.72.

2. Benutzerfreundlichkeit

Ein Benutzer soll mit wenig theoretischen Vorkenntnissen in der Lage sein, für seine Stichprobe eine oder mehrere Grammatiken erzeugen zu lassen und auf Wunsch Entscheidungshilfen für die Auswahl einer bestgeeignetsten Grammatik geliefert bekommen.

3. Vertretbarer Rechenaufwand

Die Rechenzeiten und der Speicherplatzbedarf für die Inferenz einer Grammatik müssen sich in einem für Anwendungen tragbaren Rahmen bewegen und dürfen mit wachsender Stichprobengröße nicht unverhältnismäßig stark ansteigen.

3.1.1. VORBEMERKUNGEN ZUR STICHPROBE

Am Anfang eines mit syntaktischen Methoden zu lösenden Mustererkennungsproblems steht in der Regel eine Menge von Mustern,

aus deren Beschreibungen die positive Stichprobe S^+ für einen Inferenzalgorithmus unter Beachtung der im Kapitel 2.2.1 beschriebenen Randbedingungen gewonnen werden kann.

Eine negative Stichprobe S^- hingegen ist nur aus der Erfahrung von mit dem speziellen Problem vertrauten Fachleuten rekrutierbar und müßte, nachdem hierfür i.a. kein Bildmaterial zur Verfügung steht, mühselig von Hand erstellt und dem Inferenzverfahren eingegeben werden. Darüberhinaus gibt es bisher nur bei sehr wenigen Inferenzverfahren [Wi 70, KnKn 76] Ansätze zur Berücksichtigung einer negativen Stichprobe $S^- \neq \emptyset$ mit Hilfe eines Fragen beantwortenden "Lehrers" und es ist ein nach wie vor offenes Problem, ob und wie sich diese oder andere Ansätze in den Inferenzprozeß eines vollautomatisch arbeitenden Verfahrens integrieren lassen.

Deshalb möchte ich mich in diesem Punkt der Meinung von A. Rosenfeld aus [CoRo 74]

"The best idea we had about negative samples was
just to exclude them, i.e. reject any grammar
that generates any of them"

anschließen und im folgenden davon ausgehen, daß *einem Inferenzverfahren nur positive Stichprobenelemente eingegeben werden* und evtl. vorhandene negative Stichprobenelemente erst später zur Prüfung der inferierten Grammatiken auf Kompatibilität mit der Stichprobe verwendet werden.

Im Interesse einer effizienten Verarbeitung der Graphen aus S^+ ist es sinnvoll, einige Restriktionen bezüglich der Form der Graphen einzuführen, die jedoch keinen Verlust an Darstellungsmöglichkeiten fürusterspezifische Eigenschaften mit sich bringen müssen.

So soll jedes Muster durch einen *zusammenhängenden* Graphen repräsentiert werden, weil die Erzeugung von mehreren völlig getrennten Graphen aus einem gemeinsamen Startgraphen zwar prinzipiell möglich, für die weitere Verarbeitung aber unnötig kompliziert zu handhaben wäre. Für den Fall, daß aus mehreren unverbundenen Einzelteilen bestehende Muster durch einen Graphen dargestellt werden sollen, kann man durch zusätzliche "unsichtbare Verbindungskanten" den erforderlichen Zusammenhang her-

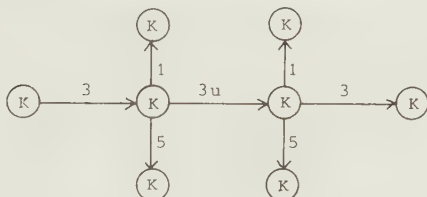
stellen.

Beispiel 5:

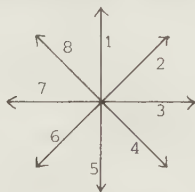
Der Kondensator



ist z.B. darstellbar durch folgenden zusammenhängenden Graphen:



wobei die Richtungen der Kanten durch Zahlen nach folgendem Schema

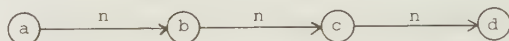


und unsichtbare Kanten mit dem Zusatz *u* gekennzeichnet sind.

Die Graphen von S^+ werden ferner als *gerichtet* vorausgesetzt, weil es in der Mustererkennung viele Arten von Relationen wie rechts von, links von, innerhalb von, Bestandteil von u.ä. gibt, die durch eine ungerichtete Kante nicht adäquat repräsentierbar sind. Es gibt zwar andererseits auch symmetrische Relationen wie verbunden mit, Nachbar von etc., die durch eine ungerichtete Kante besser darstellbar wären, aber es ist i.a. weniger schwierig, symmetrische Relationen durch gerichtete Kanten auszudrücken und darauf bei der Verarbeitung evtl. Rücksicht zu nehmen als asymmetrische Relationen mit ungerichteten Kanten darstellen zu wollen.

Beispiel 6:

Zeichenketten wie $abcd$ sind als gerichtete Graphen mit einheitlicher Kantenmarkierung



oder auch als gerichtete Graphen mit einheitlicher Knotenmarkierung

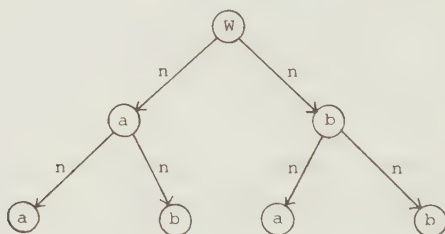


darstellbar.

Bäume wie



lassen sich am besten durch gerichtete Graphen mit einheitlich von der Wurzel zu den Blättern ausgerichteten Kanten repräsentieren:



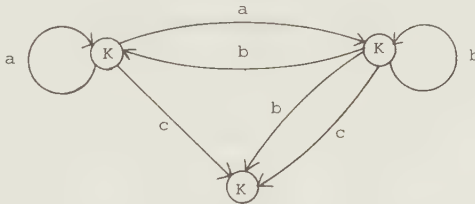
Alle in dieser Arbeit für gerichtete Graphen entwickelten Algorithmen lassen sich ohne grundsätzliche Schwierigkeiten auf den Fall ungerichteter Graphen umschreiben; es ist jedoch nicht sinnvoll, beide Fälle mit einem einzigen Programm abdecken zu wollen, weil die Datenstrukturen und Suchstrategien zu unterschiedlich dafür sind.

Eine weitere Beschränkung der Form der Graphen aus S^+ ist

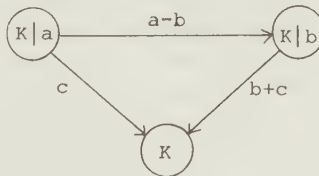
die Voraussetzung, daß die Graphen *schlingenf*rei sein müssen und *zwischen je zwei Knoten nur maximal eine Kante* enthalten dürfen. Diese beiden Restriktionen sind leicht einzuhalten, da Schlingen durch entsprechende Zusätze bei der Knotenmarkierung und Mehrfachkanten durch Zusätze bei der Kantenmarkierung darstellbar sind.

Beispiel 7:

Der Schlingen und Mehrfachkanten enthaltende Graph



ist z.B. mit der Vereinbarung, daß Schlingen hinter einem Sonderzeichen "|" bei der Knotenmarkierung und Mehrfachkanten bei gleicher Richtung mit "+" bzw. bei entgegengesetzter Richtung mit "-" bei der Kantenmarkierung angefügt werden, abbildbar auf den Graphen



ohne Schlingen und Mehrfachkanten.

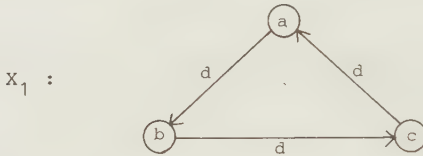
n-äre Relationen mit $n > 2$ sind in einem Graphen darstellbar entweder durch ringförmige Verkettung der *n* beteiligten Knoten oder durch Hinzunahme eines neuen Knotens, von dem aus Kanten zu den *n* Knoten führen.

Es wäre prinzipiell auch denkbar, *n*-äre Relationen nicht auf binäre Relationen zurückzuführen, sondern durch neue Arten von Verknüpfungselementen für Knoten direkt auszudrücken [Ra 73], doch würden die Vorteile dieser Vorgehensweise den

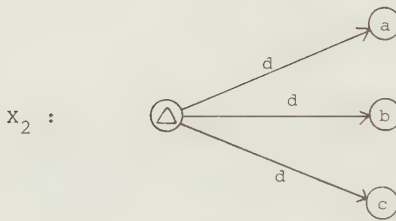
für die Behandlung dieser Verknüpfungsstrukturen erforderlichen Mehraufwand i.a. nicht rechtfertigen.

Beispiel 8:

Die Relation $d(a,b,c) \triangleq$ Dreieck mit den Ecken a, b und c ist in Graphform z.B. repräsentierbar durch den Graphen



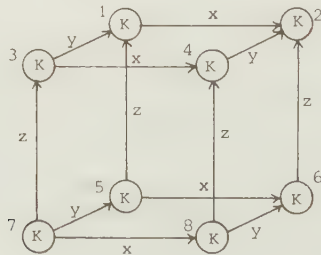
oder den Graphen



Die Graphen von S^+ brauchen *nicht planar* zu sein, wodurch sich auch komplex zusammenhängende und ursprünglich dreidimensionale Musterstrukturen problemlos darstellen lassen.

Beispiel 9:

Die Beschreibung eines Würfels in Graphform hat folgendes Aussehen:



Zum Vergleich sei die Beschreibung desselben Würfels in PDL [Sh 69] aus [Wi 78] angegeben:

$$((x*((z^i+x) + (\sim z^j))) * (((/z^1) + y) + ((x*((\sim z) + (x^k+z))) + (\sim y))) + (\sim(/z^j)))) * ((y+(/x^k)) + (\sim y)))$$

Die PDL-Beschreibung beginnt bei Knoten 7 und endet bei Knoten 8, außerdem ist die Kante von Knoten 7 zu Knoten 3 mit z^1 , die Kante (8,4) mit z^j und die Kante (5,6) mit x^k zusätzlich indiziert.

Die PDL-Beschreibung demonstriert, wie wichtig die Bemerkungen zum Thema Anschaulichkeit sein können.

3.1.2 VORBEMERKUNGEN ZUR FORM DER PRODUKTIONEN

Die Form der Produktionen der zu konstruierenden Graphgrammatiken nimmt eine Art Schlüsselstellung für die gesamte weitere Arbeit ein, weil die Güte und Verwendbarkeit der inferierten Grammatiken hiervon ganz entscheidend abhängen.

Die wichtigsten Kriterien zur Festlegung der Form der Produktionen sind:

- a) Die Form der Produktionen muß gewährleisten, daß die damit erzeugbaren Sprachen entscheidbar sind.
- b) Es müssen alle Graphstrukturen, die in einer Stichprobe vorkommen können, erzeugbar sein.
- c) Die Prüfung der Anwendbarkeit und die Anwendung einer Produktion sollen sowohl von Hand als auch per Programm leicht und schnell durchführbar sein.
- d) Die inferierten Grammatiken sollen mit möglichst wenigen möglichst einfach strukturierten Produktionen eine die Stichprobe sinnvoll verallgemeinernde Sprache erzeugen.

Forderung a) ist nach Satz 3b) relativ leicht erfüllbar durch die Festlegung, daß die *Graphproduktionen* *monoton* sein müssen.

Zur Erreichung des in Forderung b) gesteckten Ziels ist es wichtig, daß die Produktionen die Ableitung von ringförmig

strukturierten oder komplex zusammenhängenden Graphen wie z.B. dem folgenden

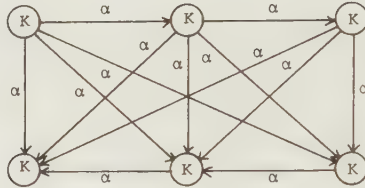


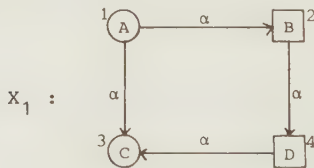
Bild 5

Beispiel für einen komplex strukturierten Graphen

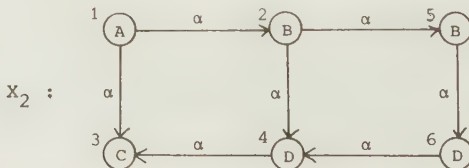
erlauben. Dies ist nach den bisherigen Erfahrungen mit verschiedenen stark eingeschränkten Produktionen und Einbettungsüberführungen entweder durch kontextfreie Produktionen in Verbindung mit einer sehr komplizierten und leistungsfähigen Einbettungsvorschrift [Na 76, LeYu 78] oder durch kontextsensitive Produktionen mit einer einfacheren Einbettungsvorschrift [Mo 70] möglich. Kontextfreie Produktionen und einfache Einbettungsüberführungen genügen dafür nicht [Pa 72, AbMiToTa 73, ViGh 78].

Beispiel 10:

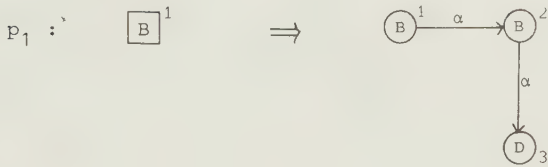
Die Ergänzung eines Graphen X_1



zu einem Graphen X_2 mit folgender Gestalt



erfordert entweder zwei kontextfreie Produktionen der Form



und



mit der zusätzlichen Vorschrift, daß eine α -Kante vom Knoten 6 in X_2 zum Knoten auf der rechten Seite der Produktion p_2 generiert werden muß,

oder eine kontextsensitive Produktion der Form



ohne zusätzliche Einbettungsvorschriften.

Dies bedeutet, daß man zur Erzeugung beliebig komplexer Graphstrukturen einen der beiden Nachteile komplizierte Einbettungsvorschrift oder nicht-kontextfreie Produktionen in Kauf nehmen muß. Aufgabenstellungen ähnlich der aus Beispiel 10 und die Beachtung von Kriterium c) legen eine Entscheidung für die *nicht-kontextfreien Produktionen* nahe, weil man diese durch Verwendung von so wenig Kontext wie möglich noch einigermaßen übersichtlich und schnell verarbeitbar halten kann.

Die nächste naheliegende Fragestellung ist nun, wie eine zunächst einmal möglichst kleine und einfach strukturierte Menge von Graphproduktionen zur Erzeugung beliebiger Stich-

probengraphen konkret aussehen muß.

Hierfür sind der Umgang mit terminalen und nichtterminalen Knoten und Kanten, die genaue Spezifikation der Einbettungsüberführung und die Struktur zulässiger rechter und linker Seiten der Graphproduktionen festzulegen.

Nachdem man es bei einer Stichprobe von Mustern mit statischen Objekten zu tun hat, die ihre Gestalt im Laufe des Verarbeitungsprozesses nicht verändern, ist es möglich und sinnvoll, die einzelnen Stichprobengraphen durch Produktionen zu erzeugen, die nicht nur bezüglich der Knoten wie in Definition 11a) gefordert, sondern auch *bezüglich der Kanten monoton* sind.

Die Kanten eines Stichprobengraphen dienen zur Darstellung von statischen Relationen zwischen je zwei Knoten, die im Verlauf des Erzeugungsprozesses eines Graphen erhalten bleiben. Infolgedessen gibt es keinen zwingenden Grund für die Einführung eines nichtterminalen Kantenmarkierungsalphabets und es sollen im folgenden *nur noch terminale Kantenmarkierungen* verwendet werden. Die Knoten eines Graphen müssen unterschiedlich nichtterminal und terminal markierbar sein, wenn man Ansatzknoten für weitere Produktionen kennzeichnen und das Ende eines Ableitungsprozesses erkennen können will. Nachdem dies für die syntaktische Mustererkennung wichtige Gesichtspunkte sind, sollen *Knoten sowohl terminal als auch nichtterminal markierbar* sein. Zusätzlich wird vereinbart, daß die Menge der in einen terminalen Knoten ein- und auslaufenden Kanten durch keine Produktion mehr verändert werden darf.

Die Einbettungsvorschrift soll im Hinblick auf Kriterium c) möglichst mit einem Blick erfaßbar und effizient verarbeitbar sein, wozu die Maxime "soviel Kontext wie nötig, sowenig Kontext wie möglich" entscheidend beiträgt.

Ein erster Schritt zur Verwirklichung dieses Grundsatzes ist die Reduzierung des Umfangs der rechten und linken Seite einer Produktion auf diejenigen Knoten und Kanten, die von der Anwendung der Produktion unmittelbar betroffen sind. Dies bedeutet einen Verzicht auf die Mitführung von Kanten, die durch die Anwendung der Produktion unverändert bleiben und die nur

der Untergraphen-Bedingung aus Definition 10 wegen bisher nicht fehlen durften.

Beispiel 11:

Die kontextsensitive Produktion p_3 aus Beispiel 10 würde dadurch folgende Form erhalten:



und enthielte nur noch Knoten, deren ein- oder auslaufende Kanten ergänzt werden, sowie neu hinzukommende Knoten und Kanten.

Um die Voraussetzung aus Kapitel 3.1.1, daß Graphen aus S^+ zwischen je zwei Knoten nur maximal eine Kante enthalten dürfen, nicht zu verletzen, muß man nun allerdings zusätzlich sicherstellen, daß eine Produktion nur dann anwendbar ist, wenn dadurch keine Mehrfachkanten in der Graph-Satzform entstehen.

Auf dieser immer noch etwas informalen Basis kann man nun allmählich an die Konstruktion von zunächst möglichst einfachen Produktionen zur Erzeugung beliebiger Graphstrukturen denken.

Dazu benötigt man zunächst eine Startproduktion



die einen nichtterminalen ersten Knoten eines Graphen liefert, sowie weitere Produktionen der Form



bzw.



die von einem vorhandenen nichtterminalen Knoten aus eine neue Kante zu einem neuen terminalen oder nichtterminalen Nachbarknoten erzeugen, und schließlich noch Produktionen der Form

$$S1 : \quad \boxed{A_1}^1 \quad \boxed{A_2}^2 \Rightarrow \boxed{A_1}^1 \xrightarrow{\alpha_1} \boxed{A_2}^2$$

die evtl. noch fehlende Kanten zwischen zwei vorhandenen nichtterminalen Knoten generieren.

Satz 6:

Jeder Graph, der den Voraussetzungen von Kapitel 3.1.1 genügt, ist durch Produktionen der Formen S0, K1 und S1 erzeugbar.

Beweis:

Gegeben sei ein beliebiger knoten- und kantenmarkierter, zusammenhängender, schlingenfreier, gerichteter Graph ohne Mehrfachkanten.

Die Erzeugung dieses Graphen beginnt mit einer Produktion vom Typ S0, die den Knoten mit der Bezeichnung 1 generiert:

$$\text{START} \Rightarrow \boxed{m_N \{b_N^{-1}(1)\}}^1$$

Nachdem der Graph zusammenhängend ist, gibt es nun mindestens einen Knoten i , der mit Knoten 1 durch eine Kante $(1,i)$ oder $(i,1)$ verbunden ist. Diese Kante und der Knoten i werden erzeugt durch eine Produktion vom Typ K1 der Form

$$\boxed{m_N \{b_N^{-1}(i)\}}^1 \Rightarrow \boxed{m_N \{b_N^{-1}(i)\}}^1 \xrightarrow{\alpha_1} \boxed{m_N \{b_N^{-1}(i)\}}^2$$

oder

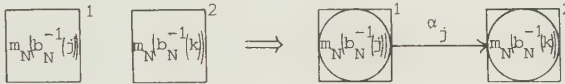
$$\boxed{m_N \{b_N^{-1}(i)\}}^1 \Rightarrow \boxed{m_N \{b_N^{-1}(i)\}}^1 \xrightarrow{\alpha_i} \boxed{m_N \{b_N^{-1}(i)\}}^2$$

wobei die Knoten der rechten Seite dann terminal werden, wenn keine ein- oder auslaufenden Kanten für diesen Knoten mehr zu erzeugen sind.

Dieses Verfahren, von erzeugten nichtterminalen Knoten aus die noch fehlenden Nachbarknoten durch Produktionen vom Typ K1 zu

generieren, entspricht der Konstruktion eines Gerüstes für den Graphen und endet genau dann, wenn alle Knoten des Graphen erzeugt sind.

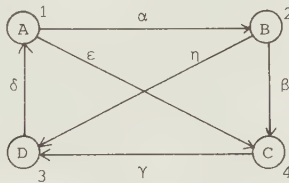
Die nun noch fehlenden Kanten (j,k) werden durch Produktionen vom Typ S1



hinzugefügt, wobei für die terminale Markierung der Knoten auf der rechten Seite der Produktion die gleiche Vorschrift wie oben gilt.

Beispiel 12:

Der Graph von Bild 1



ist erzeugbar durch die Produktionen

1. START $\Rightarrow$ $\boxed{A}$
2. $\boxed{A}^1 \Rightarrow \boxed{A}^1 \xrightarrow{\alpha} \boxed{B}^2$
3. $\boxed{B}^1 \Rightarrow \boxed{B}^1 \xrightarrow{\beta} \boxed{C}^2$
4. $\boxed{C}^1 \Rightarrow \boxed{C}^1 \xrightarrow{\gamma} \boxed{D}^2$
5. $\boxed{D}^1 \quad \boxed{A}^2 \Rightarrow \boxed{D}^1 \xrightarrow{\delta} \boxed{A}^2$
6. $\boxed{A}^1 \quad \boxed{C}^2 \Rightarrow \boxed{A}^1 \xrightarrow{\epsilon} \boxed{C}^2$
7. $\boxed{B}^1 \quad \boxed{D}^2 \Rightarrow \boxed{B}^1 \xrightarrow{\eta} \boxed{D}^2$

Das Enthaltensein der drei Produktionentypen SO , $K1$ und $S1$ in der Menge der möglichen Graphproduktionen ist nach Satz 6 in der Lage, die Erfüllung von Kriterium b) sicherzustellen. Alle drei Produktionentypen sind so einfach anwendbar, daß sie auch den Anforderungen von Kriterium c) in idealer Weise entsprechen. Nur in bezug auf Kriterium d) lassen sie noch etliches zu wünschen übrig, da Grammatiken wie die aus Beispiel 12 zu viele Produktionen enthalten und die Stichprobe zu sehr verallgemeinern. Aus diesem Grund ist es notwendig, weitere Typen von Produktionen, die größere Teilstrukturen erzeugen können, zu den bisherigen Typen hinzuzunehmen.

Das Hauptproblem hierbei ist die Tatsache, daß das Finden geeigneter Teilstrukturen, aus denen sich die Stichprobenelemente in natürlicher Weise zusammensetzen lassen, für Graphen eine wesentlich diffizilere Angelegenheit ist als beispielsweise für Zeichenketten oder Bäume, wo sich Teilzeichenketten bestimmter Länge bzw. Teilbäume bestimmter Tiefe für diesen Zweck geradezu anbieten. Bei Graphen hingegen gibt es i.a. keinen so offensichtlichen Aufbau aus Teilgraphen bestimmter Größe und die Zahl der möglichen Teilgraphen mit drei, vier oder mehr Knoten ist zu groß, um alle zu erlaubten rechten Seiten einer Graphproduktion erklären zu können. Experimente mit geschlossenen Teilstrukturen wie Dreiecken oder Vierecken haben ergeben, daß derartige Teilgraphen lediglich zur Erzeugung planarer, schwach zusammenhängender Stichprobengraphen einigermaßen nutzbringend einsetzbar sind und in dieser Richtung keine für Produktionen besonders geeigneten Teilgraphstrukturen zu finden sein werden.

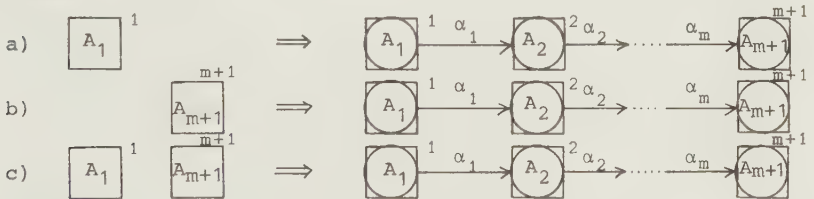
Bessere Ergebnisse liefern offene Teilstrukturen wie Kantenzüge bestimmter Länge oder Teilbäumen der Tiefe eins ähnliche sternförmige Strukturen. Erstere eignen sich zur Zusammenfassung einer Folge von $K1$ -Produktionen und sind bei fast allen Arten von Graphen gut verwendbar, letztere eignen sich zur Zusammenfassung mehrerer $S1$ -Produktionen und spielen bei der Erzeugung von Kanten in stark zusammenhängenden Graphen eine wichtige Rolle.

Diese Erfahrungen haben letztendlich zu folgenden Erweiterungen

der Form zulässiger Graphproduktionen für den Inferenzalgorithmus geführt:

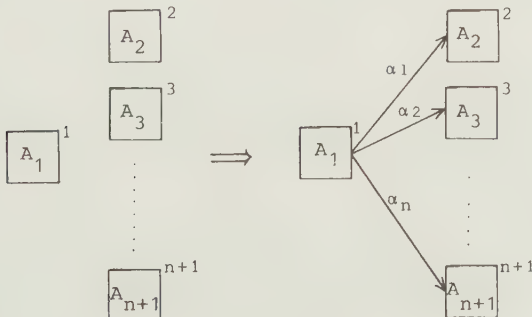
1. Die Produktionen vom Typ K1 werden dahingehend verallgemeinert, daß an einen nichtterminalen Knoten nicht nur gerichtete Kantenzüge der Länge 1, sondern gerichtete Kantenzüge der Länge n angehängt werden können. Außerdem soll es möglich sein, diese Kantenzüge nicht nur an einem, sondern auch an zwei verschiedenen nichtterminalen Knoten zu "befestigen".

Diese Produktionen sollen im weiteren *Kettenproduktionen* vom Typ K_m genannt werden und können eine der drei folgenden Formen besitzen:



2. Die Produktionen vom Typ S1 werden dahingehend verallgemeinert, daß von einem nichtterminalen Knoten aus nicht nur eine, sondern n Kanten zu anderen nichtterminalen Knoten erzeugt werden können.

Diese Produktionen sollen im weiteren *Sternproduktionen* vom Typ S_n genannt werden und folgende Form besitzen:



Die Kettenproduktionen der Formen a) und b) sind kontextfrei, die Kettenproduktionen der Form c) und die Sternproduktionen sind kontextsensitiv, wobei der *Kontext* lediglich aus einer Menge von einem bis n *nichtterminalen Knoten* besteht.

Die Abbildung der Knoten der linken Seite auf Knoten der rechten Seite ist bei der gewählten Anordnung der Graphknoten so offensichtlich, daß im folgenden auf eine Numerierung der Knoten auf der rechten und linken Seite der Graphproduktionen verzichtet werden kann.

Nach der Festlegung der Form der Produktionen ist nun die genaue Spezifikation der Ableitbarkeit als Neufassung von Definition 10 möglich:

Definition 13:

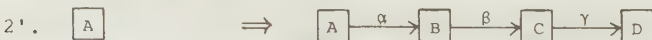
Ein Graph X_{i+1} heißt aus einem Graphen X_i mittels einer Produktion $p = (X_l, X_r)$ vom Typ SO , Km oder Sn ableitbar gdw.

- $X_l \subseteq X_i$ und für Produktionen vom Typ Sn in der Graph-Satzform X_l noch keine Kanten zwischen den Knoten A_1 und A_2 , A_1 und A_3 , ..., A_1 und A_{n+1} enthalten sind
- $X_r \subseteq X_{i+1}$ und in der Graph-Satzform vorhandene Kanten zwischen den Knoten von X_l unverändert in X_r übernommen werden
- $X_i - X_l = X_{i+1} - X_r$.

Die Sprache einer Graphgrammatik mit Produktionen der Typen SO , Km und Sn ist die Menge aller Graph-Satzformen, die aus $START$ ableitbar sind und ausschließlich terminale Knotenmarkierungen besitzen.

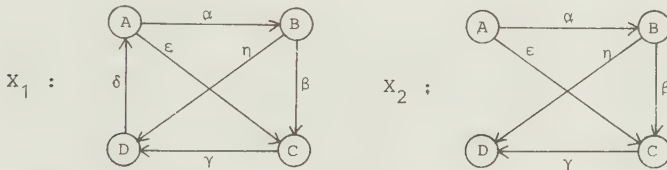
Beispiel 13:

Die Verwendung der Produktionen vom Typ Km und Sn mit $m, n > 1$ vereinfacht die Erzeugung des Graphen aus Beispiel 12 dahingehend, daß die Produktionen Nummer 2, 3 und 4 zu einer einzigen Produktion vom Typ $K3$



zusammengefaßt werden können. Bei den Sternproduktionen ist wegen der unterschiedlichen Sternausgangsknoten keine Zusammenfassung möglich. Die Menge der mit den Produktionen 1, 2', 5, 6 und 7 er-

zeugbaren terminalen Graphen reduziert sich dadurch auf die zwei Graphen X_1 und X_2



gegenüber einer unendlichen Menge erzeugbarer Graphen durch die Produktionen 1 bis 7 in Beispiel 12.

Aus dem letzten Beispiel wird erkennbar, daß die Einführung der Produktionen zur Erzeugung größerer Teilgraphen die Erfüllbarkeit von Kriterium d) wesentlich erleichtert ohne dabei der Erfüllbarkeit von Kriterium c) im Wege zu stehen.

3.1.3 VORBEMERKUNGEN ZUR INFERENZSTRATEGIE

Die gewählte Inferenzstrategie ist sowohl für die Qualität der resultierenden Grammatiken als auch für den Rechenaufwand und damit für die Praktikabilität eines Inferenzverfahrens von grundlegender Bedeutung.

Zur Festlegung der Strategie eines Inferenzverfahrens müssen verschiedene Entscheidungen über Vorgehensweisen zur Gewinnung und Prüfung der Grammatiken getroffen werden, wobei man immer das Ziel vor Augen hat, mit möglichst wenig Aufwand eine möglichst gute Grammatik zu erhalten.

Die erste in diesem Zusammenhang anstehende Entscheidung ist die für einen enumerativen oder für einen konstruktiven Ansatz zur Inferenz von Graphgrammatiken. Aufgrund der schon in Kapitel 1 erwähnten Rechenzeiten der enumerativen Verfahren für weit einfacher handhabbare und weniger variable Grammatiken muß man den enumerativen Ansatz als für die vorliegende Problemstellung derzeit absolut impraktikabel einstufen und sich für einen *konstruktiven Ansatz* entscheiden.

Für den Zugriff auf die Stichprobenelemente gibt es unter den konstruktiven Verfahren im wesentlichen zwei verschiedene Konzepte:

- a) Das erste und häufiger gebrauchte Konzept geht davon aus, daß es möglich und nötig ist, ständig die gesamte Stichprobe verfügbar zu halten und nutzt dies zur beliebig häufigen Suche nach in der Stichprobe enthaltenen Unterstrukturen aus.
- b) Das zweite, als inkrementelle Vorgehensweise bekannte Konzept geht davon aus, daß man jedes Stichprobenelement einzeln untersuchen und im Anschluß daran die Grammatik bei Bedarf entsprechend erweitern kann.

Eine Anwendung von Konzept a) würde bedeuten, daß alle Graphen einer Stichprobe zusammen mit allen vorkommenden Teilgraphen bis zu einer bestimmten Größe ständig im Arbeitsspeicher verfügbar sein müßten, und daß auf der Suche nach Unterstrukturen eine Vielzahl von äußerst rechenintensiven Unter- bzw. Teilgraphentests durchzuführen wären. Bei einer Entscheidung für die inkrementelle Vorgehensweise müßte hingegen immer nur ein Stichprobenelement und die aktuelle Zwischengrammatik verfügbar sein, was selbst bei kleineren Rechenanlagen noch ohne Schwierigkeiten möglich ist.

Der Vorteil des ersten Konzepts besteht hauptsächlich darin, daß man die in der Stichprobe am häufigsten auftretenden Teilstrukturen ermitteln und durch Erzeugung dieser Teilstrukturen mit jeweils einer Produktion zu einer relativ kurzen Grammatik für die Stichprobe gelangen kann. Die Erfahrungen mit dem nach diesem Prinzip arbeitenden Cook'schen Inferenzverfahren für kontextfreie Zeichenketten zeigen, daß diese Vorgehensweise durchaus vielversprechende Resultate liefert, daß man aber bei Stichproben von mehreren hundert Zeichenketten bereits an die Grenzen der normalerweise zur Verfügung stehenden Rechnerkapazität stoßen kann und daß die Programmlaufzeiten mit dem Umfang der Stichprobe überproportional stark ansteigen [Du 79]. Nachdem zu erwarten ist, daß diese Grenzen für Stichproben von Graphen viel früher erreicht werden als bei Zeichenketten und es außerdem auch bei inkrementeller Vorgehensweise Möglichkeiten dafür gibt, häufig in der Stichprobe vorkommende Teil-

strukturen durch wenige Produktionen zu modellieren, erscheint der für eine Realisierung von Konzept a) zu zahlende Preis unangemessen hoch und eine *Entscheidung für Konzept b)* die vernünftigeren Lösung zu sein.

Dies bietet obendrein noch den Vorteil, daß das Inferenzverfahren bei nachträglicher Vergrößerung der Stichprobe mit der Konstruktion einer Grammatik nicht wie bei Konzept a) noch einmal ganz von vorne anfangen muß, sondern ähnlich dem Prinzip des "open-ended design" ohne großen Aufwand in der Lage ist, die Erzeugbarkeit neuer Stichprobenelemente in die Grammatik einzubauen. Auch die Rechenzeiten für die Inferenz einer Grammatik werden bei einer Realisierung von Konzept b) ungefähr proportional zur Stichprobengröße ansteigen, was bei großen Stichproben einen beachtlichen Vorteil gegenüber Konzept a) darstellt.

Eine weitere konzeptionelle Entscheidung ist die Frage, ob das Inferenzverfahren völlig autonom nach dem Schema Stichprobeneingabe - Grammatikausgabe oder in Interaktion mit einem den Inferenzprozeß steuernden und gegebenenfalls abbrechenden Benutzer arbeiten soll. Die zweite Alternative ist allerdings nur dann sinnvoll einsetzbar, wenn das Verfahren in der Lage ist, einem Benutzer soviel Zwischenergebnisse anzubieten, wie er braucht, um über die Fortsetzung des Verfahrens entscheiden zu können, und wenn außerdem die Rechenzeiten zwischen zwei Benutzerreaktionen nicht ermüdend lang ausfallen. Nachdem die erste Voraussetzung durch Beobachtung der Veränderungen an der aktuellen Zwischengrammatik und die zweite Voraussetzung durch Niedrighalten der Verarbeitungszeit für ein Stichprobenelement erfüllbar sind, sollen dem Benutzer gewisse *Interaktionsmöglichkeiten* zum Zwecke einer schnelleren und gezielteren Suche nach einer bestgeeignetsten Grammatik eingeräumt werden können, die sich aber im wesentlichen darauf beschränken, die Konstruktion wenig erfolgversprechender Grammatiken möglichst frühzeitig abubrechen und die den ansonsten vollautomatischen Ablauf des Verfahrens nicht beeinträchtigen.

3.2 INFERENZVERFAHREN

Dieses Kapitel beschreibt die Realisierung und die Ergebnisse zweier Inferenzverfahren, die sich lediglich in der Ausnutzung der in der aktuellen Zwischengrammatik enthaltenen Information unterscheiden. Deshalb wird das erste Verfahren in seiner Gesamtkonzeption vorgestellt und beim zweiten Verfahren nur über die gegenüber dem ersten Verfahren vorgenommenen Änderungen und deren Auswirkungen berichtet.

3.2.1 INFERENZ OHNE GEDÄCHTNIS

Die Kontrollstruktur eines inkrementellen Inferenzverfahrens läßt sich am besten anhand des folgenden Ablaufdiagramms erläutern:

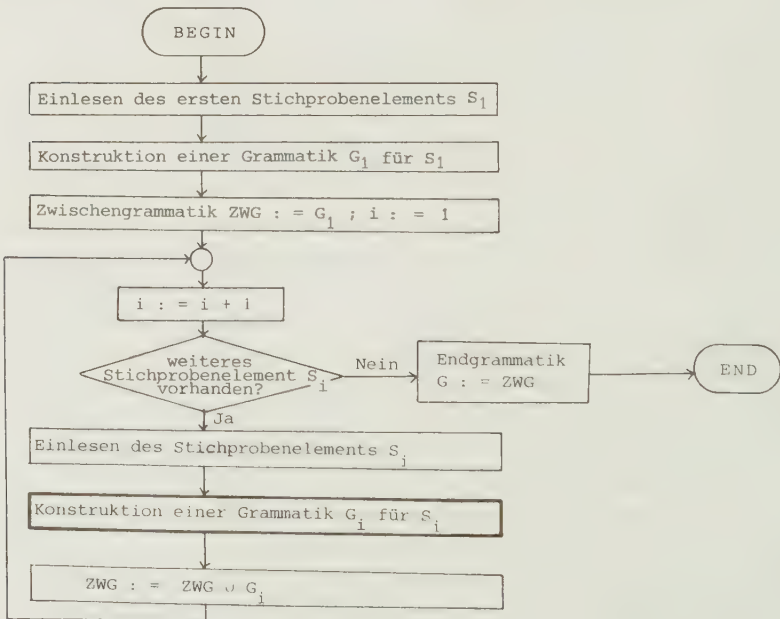


Bild 6

Kontrollstruktur eines inkrementellen Inferenzverfahrens

Das Inferenzverfahren beginnt mit dem Einlesen eines ersten Stichprobenelements S_1 . Für S_1 wird eine Grammatik G_1 konstruiert, die die Bedingung $S_1 \in L(G_1)$ erfüllt. Die aktuelle Zwischengrammatik ZWG, die ein Modell für die Struktur der bisher verarbeiteten Muster darstellt, wird mit der Grammatik G_1 gleichgesetzt.

Für alle weiteren Stichprobenelemente wird nun ein Zyklus aus drei verschiedenen Programmteilen durchlaufen, von denen der erste das Stichprobenelement S_i einliest, der zweite eine Grammatik G_i mit der Eigenschaft $S_i \in L(G_i)$ konstruiert und der dritte die Grammatik ZWG durch Vereinigung mit G_i dem neuen Stichprobenelement anpaßt.

Falls alle Stichprobenelemente wie beschrieben abgearbeitet sind, wird die letzte aktuelle Zwischengrammatik als Endgrammatik G ausgegeben und das Inferenzverfahren beendet.

Der einzige Programmteil, in dem sich die beiden Inferenzverfahren ohne bzw. mit Gedächtnis unterscheiden, ist der stark umrandete Block "Konstruktion einer Grammatik G_i für S_i ", der als das Kernstück des Inferenzverfahrens angesehen werden kann.

3.2.1.1 DARSTELLUNG UND BEARBEITUNG DER STICHPROBENELEMENTE

Für die Darstellung von Graphen im Computer gibt es eine Vielzahl von Möglichkeiten [De 74], die für die Repräsentation der in Kapitel 2.1.1 beschriebenen Stichprobenelemente und die im Laufe des Inferenzprogramms durchzuführenden Operationen zum Suchen und Ändern von Teilgraphen unterschiedlich gut geeignet sind.

Wichtig sind im vorliegenden Fall Datenstrukturen, die die Suche nach teilweise nur unvollständig spezifizierten Teilgraphen begünstigen und schnelle Zugriffe von den Bezeichnungen der Knoten und Kanten auf ihre Markierungen und umgekehrt erlauben. Weniger wichtig sind Möglichkeiten zum Entfernen oder Hinzufügen von Teilgraphen, da die Graphen der Stichprobe ihre Form nicht verändern und abgearbeitete Teil-

graphen statt gelöscht auch entsprechend gekennzeichnet werden können.

Die endgültige Form der Datenstrukturen für die Graphen ist natürlich auch abhängig von den Möglichkeiten, die die zur Implementierung des Inferenzalgorithmus gewählte Programmiersprache zur Darstellung von Graphstrukturen bereithält. Generell von Vorteil für den Umgang mit Stichprobenelementen der beschriebenen Art sind elementare Datentypen der Form Character oder String zur Aufnahme der Knoten- und Kantenmarkierungen sowie ein Befehl zur Feststellung der Identität zweier Markierungen. Die Formulierung von Suchalgorithmen läßt sich erleichtern durch ein Pointer-Konzept, und zum Zusammenpacken aller für eine Produktion wichtigen Daten sind aus unterschiedlichen Datentypen zusammensetzbare Records nützlich.

Nachdem aber zu Beginn dieser Arbeit die Forderung bestand, die Inferenzprogramme sollten auf der lehrstuhleigenen Rechanlage vom Typ PDP 11/34 installierbar sein, kam für die Implementierung der Inferenzalgorithmen nur FORTRAN als einzige auf der PDP verfügbare höhere Programmiersprache in Frage und das bedeutete, daß die *Graphen der Stichprobe in Feldern untergebracht* werden mußten.

Hierfür benötigt man zunächst einmal eine *Liste für die Kanten* des Graphen, die mindestens die Quellknotennummer, die Zielknotennummer und die Markierung jeder Kante enthält. Zur vervollständigung der Graphdaten könnte man in der Kantenliste noch zu jedem Quell- bzw. Zielknoten seine Markierung dazupacken, aber diese Datenstruktur enthielte dann viele Knotenmarkierungen mehrfach und wäre umständlich zu bearbeiten. Besser ist deshalb statt der expliziten Knotenmarkierung ein Verweis auf einen Eintrag in einer gesonderten *Knotenliste*, die für jeden Knoten seine Nummer, seine Markierung und zur Vereinfachung der Algorithmen evtl. noch die Zahl seiner ein- und auslaufenden Kanten enthält.

Beispiel 14:

Der Graph aus Bild 1 wird durch folgende Datenstruktur dargestellt:

Kantenliste

Quell-knoten	Ziel-knoten	Markierung
1	2	ALPHA
1	3	EPSILON
2	3	BETA
2	4	ZETA
3	4	GAMMA
4	1	DELTA

Knotenliste

Nummer	Markierung	Anz. einl.K.	Anz. ausl.K.
1	A	1	2
2	B	1	2
3	C	2	1
4	D	2	1

Dieser Exkurs über normalerweise unwichtige Details der Programmierung ist notwendig, um verständlich zu machen, was in den Inferenzalgorithmen der "nächste Knoten" oder die "nächste Kante" in der entsprechenden Liste bedeuten. Außerdem sind unterschiedliche Ergebnisse aufgrund von Variationen der Reihenfolge der Knoten und Kanten in der Datenstruktur ohne diesen Hintergrund nicht erklärbar.

Die Bearbeitung der Graphen verlangt vielfach die Suche nach teilweise unvollständig spezifizierten Teilgraphen mit bestimmten Eigenschaften. Für den Fall, daß ein Graph nur einen Teilgraphen mit der gesuchten Eigenschaft enthält, ist das Ergebnis der Suche von der Reihenfolge der Speicherung der Knoten und Kanten unabhängig. Falls der Graph aber mehrere Teilgraphen mit der gewünschten Eigenschaft enthält, muß einer davon für die vorgesehene Aktion ausgewählt werden und die Inferenzalgorithmen sind so angelegt, daß dies i.a. der zuerst gefundene sein wird.

Hieraus wird ersichtlich, daß die Knoten- und Kantenlisten zwar keinen Ersatz für die bei Graphen fehlende Abarbeitungsrichtung wie z.B. von links nach rechts bei Zeichenketten oder breadth-first bei Bäumen bieten, jedoch in gewissen Situationen eine Entscheidung zugunsten eines bestimmten Ansatzknotens o.ä. herbeiführen können.

3.2.1.2 KONSTRUKTION EINER EINZELGRAMMATIK

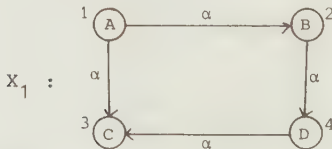
Das Ziel bei der Konstruktion einer Einzelgrammatik G_1 für ein Stichprobenelement S_1 besteht im wesentlichen darin, S_1 möglichst geschickt mit möglichst wenigen Ketten- und Sternproduktionen zu erzeugen.

Das erste Problem hierbei ist die Auswahl des Startknotens, der die Form der *Startproduktion* festlegt. Dieser Knoten sollte in gewisser Weise charakteristisch für eine größere Menge von ähnlichen Mustern sein, weshalb eine zufällige Auswahl oder eine Auswahl des ersten Knotens in der Knotenliste keine zufriedenstellende Lösung darstellt. Außerdem wäre es für Anwendungen wie z.B. die Erkennung von Werkstücken in beliebiger Lage vorteilhaft, wenn dieser Startknoten möglichst rotationsinvariant, d.h. von der speziellen Winkellage des Musters und damit von der Reihenfolge der Knoten in der Knotenliste unabhängig wäre.

Da dieser Knoten außerdem der nichtterminale Knoten auf der linken Seite der ersten Kettenproduktion sein wird, liegt es nahe, hierfür den Anfangsknoten der längsten im Graphen vorhandenen Bahn auszuwählen. Für den Fall, daß es zwei oder mehr gleich lange längste Bahnen mit verschiedenen Anfangsknoten gibt, wird der in der Knotenliste am weitesten oben stehende ausgewählt.

Beispiel 15:

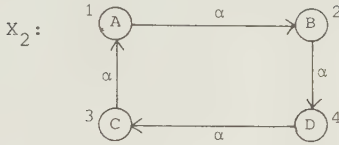
Die Erzeugung des Graphen



beginnt unabhängig von der Reihenfolge der Knoten in der Knotenliste mit der Startproduktion

START $\Rightarrow$ A .

Die Erzeugung des Graphen



beginnt, nachdem alle vier Knoten gleichermaßen als Ausgangsknoten für längste Bahnen in Frage kommen, mit der Generierung des ersten Knotens in der Knotenliste, d.h. in diesem Fall auch mit der Produktion

START $\Rightarrow$ A .

Die Auswahlvorschrift, im Zweifelsfalle mit dem in der Knotenliste am weitesten oben stehenden Knoten zu beginnen, kann je nach Vorgehensweise bei der Vorverarbeitung der Muster so interpretiert werden, daß man sich z.B. bei zeilenweiser Verarbeitung von oben nach unten und innerhalb einer Zeile von links nach rechts für den am weitesten oben links gefundenen unter den möglichen Anfangsknoten entscheidet.

Der nächste Schritt besteht nun in der Erzeugung aller im Graphen enthaltenen k Knoten sowie von mindestens $k - 1$ Verbindungskanten durch eine möglichst geringe Zahl von *Kettenproduktionen*.

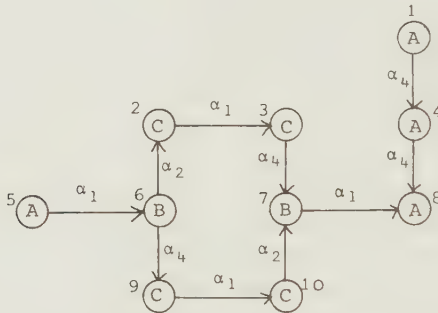
Hierfür wird zunächst die für die Ermittlung des Anfangsknotens benötigte längste Bahn im Graphen durch eine Kettenproduktion der Form a) generiert. Anschließend wird solange nach der jeweils längsten Bahn im Graphen, deren Anfangs- und/oder Endknoten bereits erzeugt und deren Zwischenknoten noch nicht erzeugt sind, weitergesucht und diese durch eine Kettenproduktion der Form a), b) oder c) generiert, bis alle im Graphen enthaltenen Knoten erfaßt sind.

Die Form der linken Seite der Kettenproduktionen richtet sich danach, ob nur der Anfangsknoten (Form a)), nur der Endknoten (Form b)) oder Anfangs- und Endknoten (Form c)) bereits er-

zeugt sind. Die rechte Seite einer Kettenproduktion der Form a) oder b) enthält mindestens eine Kante, die rechte Seite einer Kettenproduktion der Form c) mindestens zwei Kanten, so daß durch jede Kettenproduktion wenigstens ein neuer Knoten zu den bisher generierten hinzukommt.

Beispiel 16:

Der Graph



wird durch folgende Produktionen erzeugt:

Der Ausgangspunkt der längsten Bahn ist der Knoten 5, demzufolge lautet die Startproduktion:

$$1. \text{ START} \Rightarrow \boxed{A}$$

Die Kettenproduktion zur Erzeugung der längsten Bahn ist:

$$2. \boxed{A} \Rightarrow A \xrightarrow{\alpha_1} B \xrightarrow{\alpha_2} C \xrightarrow{\alpha_1} C \xrightarrow{\alpha_4} B \xrightarrow{\alpha_1} A$$

Die Kettenproduktion zur Generierung der nächstlängsten Bahn hat die Gestalt

$$3. \boxed{B} \quad \boxed{B} \Rightarrow B \xrightarrow{\alpha_4} C \xrightarrow{\alpha_1} C \xrightarrow{\alpha_2} B$$

und schließlich fehlt noch eine Kettenproduktion der Form

$$4. \boxed{A} \Rightarrow A \xrightarrow{\alpha_4} A \xrightarrow{\alpha_4} A$$

zur Erzeugung der restlichen Graphknoten.

Die ursprüngliche Idee bei der Einführung der Kettenproduktionen war, ein Gerüst des Graphen mit allen k Knoten und genau $k - 1$ Kanten durch möglichst wenige Kettenproduktionen zu erzeugen. Bei Graphen, die ein gerichtetes Gerüst in Form eines Baumes besitzen, ist dies allein durch Kettenproduktionen der Form a) möglich, bei Graphen mit ungerichtetem Gerüst benötigt man dafür auch Kettenproduktionen der Form b).

Die Experimente mit diesem "Gerüst-Konzept" lieferten jedoch keine zufriedenstellenden Grammatiken, weil alle Bahnen der Länge 1, die über einen Anfangs- und einen Endpunkt mit dem übrigen Graphen verbunden sind, nicht mit einer Kettenproduktion vom Typ K1, sondern nur mit einer Kettenproduktion vom Typ K1-1 und einer Sternproduktion vom Typ S1 erzeugbar waren.

Beispiel 17:

Die Produktion 3 aus Beispiel 16 wäre bei Beibehaltung des "Gerüst-Konzepts" zerlegt worden in die zwei Produktionen

$$3a. \quad \boxed{B} \quad \Rightarrow \quad \textcircled{B} \xrightarrow{\alpha_4} \textcircled{C} \xrightarrow{\alpha_1} \boxed{C}$$

und

$$3b. \quad \boxed{C} \quad \boxed{B} \quad \Rightarrow \quad \textcircled{C} \xrightarrow{\alpha_2} \textcircled{B}$$

wodurch sich in diesem Beispiel nur die Anzahl der Produktionen vergrößert, i.a. aber auch die Anzahl der durch die Grammatik erzeugbaren terminalen Graphen erhöhen kann.

Diese beiden unerwünschten Effekte lassen sich beseitigen durch Verwendung von Kettenproduktionen der Form c), die besonders zur Vervollständigung von zyklischen Strukturen und zur Modellierung von "Zuwachs" an bereits halbfertig aufgebauten Mustern nutzbringend einsetzbar sind.

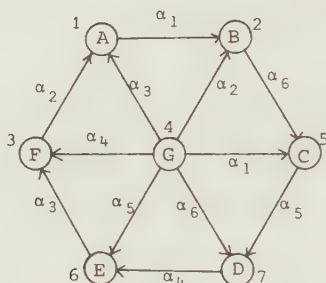
Der Suchaufwand für eine Kettenproduktion ist für die erste Kettenproduktion abhängig von der Anzahl der Knoten und Kanten des Graphen relativ groß, nimmt aber mit sinkender Anzahl noch nicht erzeugter Graphknoten ziemlich rasch ab.

Im letzten Schritt erfolgt die Erzeugung der noch fehlenden Graphkanten durch eine möglichst geringe Zahl von *Sternproduktionen*.

Zu diesem Zweck muß für jeden Graphknoten, der n noch nicht erfaßte auslaufende Kanten besitzt, eine Sternproduktion vom Typ S_n zur Erzeugung dieser Kanten konstruiert werden. Der Kontext auf der linken Seite dieser Produktionen besteht aus einem Sternausgangsknoten und sämtlichen Zielknoten der zu erzeugenden Kanten.

Beispiel 18:

Der Graph



ist erzeugbar durch die Startproduktion

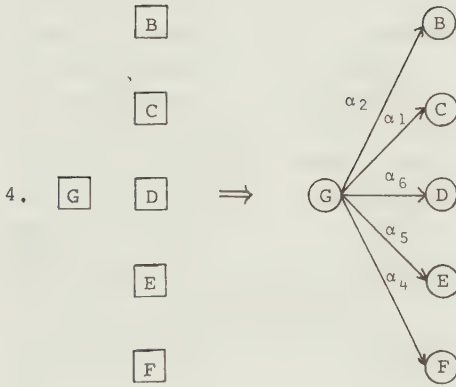
1. $\text{START} \Rightarrow \boxed{G}$,

die Kettenproduktion

2. $\boxed{G} \Rightarrow \boxed{G} \xrightarrow{\alpha_3} \boxed{A} \xrightarrow{\alpha_1} \boxed{B} \xrightarrow{\alpha_6} \boxed{C} \xrightarrow{\alpha_5} \boxed{D} \xrightarrow{\alpha_4} \boxed{E} \xrightarrow{\alpha_3} \boxed{F}$

und die beiden Sternproduktionen

3. $\boxed{F} \quad \boxed{A} \Rightarrow \boxed{F} \xrightarrow{\alpha_2} \textcircled{A}$



Die den Gesamteindruck etwas störende Produktion 3 ist erforderlich, weil die Produktion 2 keinen Knoten zweimal berühren darf und die Kante von Knoten 3 zu Knoten 1 infolgedessen durch eine gesonderte Produktion erzeugt werden muß.

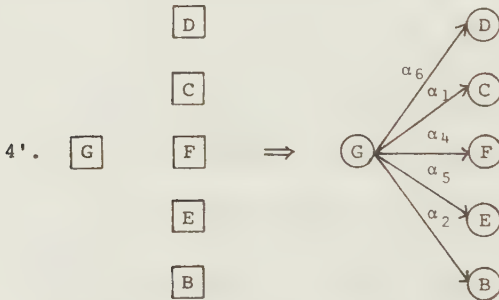
Die Reihenfolge, in der die einzelnen Sternproduktionen konstruiert werden, ist ohne Einfluß auf die Anzahl der erforderlichen Sternproduktionen, weil diese sich allein an der Anzahl der Graphknoten mit noch nicht erzeugten auslaufenden Kanten bemißt. Die Art und Weise, in der die Graphknoten durch die Sternproduktionen terminal werden, sorgt jedoch dafür, daß zur Erzeugung und auch zur Syntaxanalyse eines Graphen die Reihenfolge der anzuwendenden Sternproduktionen nur in gewissen Grenzen variabel ist.

Die Strategie, nach der der Sternausgangsknoten für die als nächstes zu konstruierende Sternproduktion ausgewählt wird, ist somit weder für den Umfang noch für die Güte der inferierten Grammatik von Bedeutung, weshalb sich die Wahl einer möglichst einfach realisierbaren Strategie empfiehlt. Bei den implementierten Inferenzalgorithmen wird die Kantenliste von vorne nach hinten nach noch nicht erfaßten Kanten abgesucht und der Quellknoten der jeweils nächsten noch zu erzeugenden Kante als Sternausgangsknoten für die nächste zu konstruierende Stern-

produktion verwendet.

Je nach Vorgehensweise bei der Erstellung der Kantenliste aus den Ergebnissen der Mustervorverarbeitung kann dieses Auswahlverfahren z.B. so interpretiert werden, daß der zuerst gefundene oder der am weitesten oben links stehende unter den noch zu verarbeitenden Sternausgangsknoten den Mittelpunkt der nächsten Sternproduktion bildet.

Die Reihenfolge, in der die Zielknoten der Sternkanten auf der linken und rechten Seite einer Produktion angeordnet sind, ist zufällig und für die Anwendbarkeit oder Wirkungsweise einer Produktion ohne Bedeutung. Demzufolge ist die Produktion 4 aus Beispiel 18 äquivalent mit der Produktion



und allen weiteren Produktionen, die sich von der vorangegangenen nur durch eine andere Reihenfolge der Zielknotenanzahl unterscheiden.

Der Suchaufwand für eine Sternproduktion ist weitgehend unabhängig von der Anzahl bereits konstruierter Sternproduktionen und der Anzahl der zu erzeugenden Kanten. Im Vergleich zum Aufwand für die Konstruktion einer Kettenproduktion ist er deutlich geringer.

3.2.1.3 VEREINIGUNG DER EINZELGRAMMATIKEN

Um aus den vielen Einzelgrammatiken G_i mit der Eigenschaft $S_i \in L(G_i)$ eine Lösungsgrammatik G für das Inferenzproblem mit der Eigenschaft $S^+ \subseteq L(G)$ zu erhalten, muß man die ver-

schiedenen Einzelgrammatiken G_i zu einer Gesamtgrammatik G zusammenfassen.

Die inkrementelle Vorgehensweise bedingt, daß diese Zusammenfassung nicht in einem Zug, sondern schrittweise abwechselnd mit der Konstruktion neuer Einzelgrammatiken während des gesamten Inferenzprozesses stattfindet. Dadurch reduziert sich das Problem auf die wiederholt auszuführende Vereinigung einer aktuellen Zwischengrammatik ZWG mit einer Einzelgrammatik G_i für das zuletzt eingelesene Stichprobenelement S_i .

Die Vereinigung dieser beiden Grammatiken erfolgt durch Hinzunahme aller in ZWG bisher noch nicht enthaltenen Produktionen aus G_i verbunden mit der Aufnahme neuer terminaler bzw. nicht-terminaler Markierungen in die entsprechenden Alphabete.

Formal läßt sich dies wie folgt definieren:

Definition 14:

Die Vereinigung von zwei Graphgrammatiken $G_1 = (V_{N1}, V_{E1}, V_{NT1}, V_{ET1}, \text{START}, P_1)$ und $G_2 = (V_{N2}, V_{E2}, V_{NT2}, V_{ET2}, \text{START}, P_2)$ zu einer dritten Graphgrammatik $G_3 = (V_{N3}, V_{E3}, V_{NT3}, V_{ET3}, \text{START}, P_3)$ (abgekürzt $G_3 := G_1 \cup G_2$) ist zurückführbar auf die Vereinigung folgender Mengen:

- $V_{N3} := V_{N1} \cup V_{N2}$
- $V_{E3} := V_{E1} \cup V_{E2}$
- $V_{NT3} := V_{NT1} \cup V_{NT2}$
- $V_{ET3} := V_{ET1} \cup V_{ET2}$
- $P_3 := P_1 \cup P_2$

Hieraus ist ableitbar:

Satz 7:

Seien $G_1 = (V_{N1}, V_{E1}, V_{NT1}, V_{ET1}, \text{START}, P_1)$ und $G_2 = (V_{N2}, V_{E2}, V_{NT2}, V_{ET2}, \text{START}, P_2)$ zwei Graphgrammatiken, deren Produktionen den Vorschriften aus Kapitel 3.1.2 genügen.

Dann trifft dies auch für die Grammatik $G_3 := G_1 \cup G_2$ zu und es

gilt:

$$L(G_3) \subseteq L(G_1) \cup L(G_2) .$$

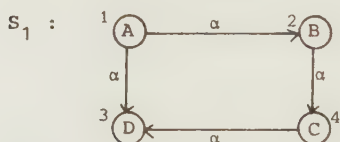
Zum Beweis:

Aus Definition 14 ist unmittelbar ersichtlich, daß sich an der Form der Produktionen durch die Vereinigung nichts ändert, weshalb auch G_3 den Vorschriften aus Kapitel 3.1.2 genügt. Außerdem sind in G_3 alle in G_1 oder G_2 ableitbaren terminalen Graphen weiterhin erzeugbar, weil die dafür erforderlichen Produktionen allesamt in P_3 enthalten sind.

An dieser Stelle sei darauf hingewiesen, daß es auch Inferenzverfahren gibt, die bei der Vereinigung von Einzelgrammatiken Umformungen der Gestalt der Produktionen mit dem Ziel einer Reduzierung der Produktionenzahl und dem Effekt einer Verallgemeinerung der erzeugbaren Sprache vornehmen [Ev 71]. Bei dem geschilderten Inferenzverfahren wurde auf derartige Umformungen verzichtet, weil eine weitere Verallgemeinerung der durch die inferierten Graphgrammatiken erzeugbaren Sprachen nicht wünschenswert erschien.

Beispiel 19:

Das Stichprobenelement



liefert die folgende Einzelgrammatik G_1

1. **START** $\Rightarrow$ A
2. A $\Rightarrow$ A $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ C $\xrightarrow{\alpha}$ D
3. A D $\Rightarrow$ A $\xrightarrow{\alpha}$ D

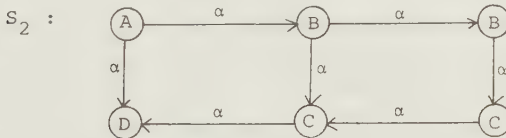
die zur ersten aktuellen Zwischengrammatik ZWG des Inferenzprozesses wird.

Durch die Grammatik G_1 ist trotz der linksrekursiven Produktion 2 nur der Graph S_1 erzeugbar. Für den Fall, daß durch die mögliche Rekursivität der ersten Kettenproduktion eine unerwünschte Verallgemeinerung der erzeugbaren Graphsprache entsteht, könnte man die ersten beiden Produktionen zu einer Produktion der Form

START $\Rightarrow$ Längste Kette im Graphen

zusammenziehen oder eine programmierte Graphgrammatik (siehe Kapitel 5) verwenden.

Die Hinzunahme des Stichprobenelements



liefert die Einzelgrammatik G_2

1. START $\Rightarrow$ A
2. A $\Rightarrow$ A $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ C $\xrightarrow{\alpha}$ C $\xrightarrow{\alpha}$ D
3. A D $\Rightarrow$ A $\xrightarrow{\alpha}$ D
4. B C $\Rightarrow$ B $\xrightarrow{\alpha}$ C

deren Produktionen 1 und 3 bereits in ZWG enthalten sind. Die Vereinigung der Grammatiken ZWG und G_2 hat deshalb folgende Produktionsmenge als Ergebnis:

1. START $\Rightarrow$ 
2.  $\Rightarrow$  $\xrightarrow{\alpha}$  $\xrightarrow{\alpha}$  $\xrightarrow{\alpha}$ 
3.  $\Rightarrow$  $\xrightarrow{\alpha}$  $\xrightarrow{\alpha}$  $\xrightarrow{\alpha}$  $\xrightarrow{\alpha}$  $\xrightarrow{\alpha}$ 
4.   $\Rightarrow$  $\xrightarrow{\alpha}$ 
5.   $\Rightarrow$  $\xrightarrow{\alpha}$ 

Die Implementierung der Operation $ZWG := ZWG \cup G_i$ erfordert für jede in G_i enthaltene Produktion die Feststellung, ob sie mit einer in ZWG enthaltenen Produktion übereinstimmt und falls dies nicht der Fall ist, die Hinzunahme dieser Produktion zu ZWG .

Bei Startproduktionen und Kettenproduktionen ist die Übereinstimmung durch einen einfachen Vergleich der rechten und linken Seiten der Produktion überprüfbar; bei Sternproduktionen muß man berücksichtigen, daß nur die Menge, nicht aber die Reihenfolge der zu erzeugenden Kanten und dazugehörigen Zielknoten übereinstimmen muß, wodurch sich der Vergleich etwas komplizierter gestaltet.

Der Rechenaufwand für die Vereinigung einer Zwischengrammatik ZWG mit einer Einzelgrammatik G_i steigt sowohl mit dem Umfang der Zwischengrammatik als auch mit dem Umfang von G_i etwa linear an, bleibt jedoch im Normalfall klein im Verhältnis zum Rechenaufwand für die Konstruktion einer Einzelgrammatik.

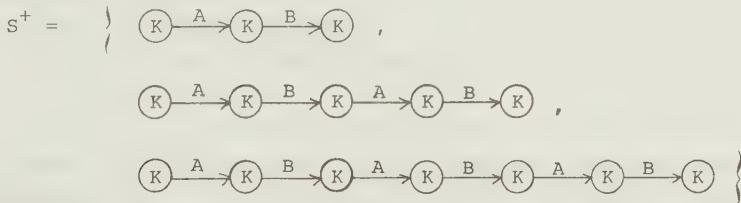
3.2.1.4 PARAMETERGESTEUERTE GRAMMATIKKONSTRUKTION

Das in Kapitel 3.2.1.2 beschriebene Verfahren zur Konstruktion einer Einzelgrammatik ist in erster Linie dafür geeignet, für einen einzelnen Graphen der Stichprobe eine Grammatik mit einer minimalen Anzahl von Ketten- und Sternproduktionen zu generieren. Bei der Anwendung dieses Verfahrens auf größere Stichproben von ähnlichen Mustern erweist es sich jedoch als äußerst nachteilig,

daß gemeinsame Teilstrukturen nur selten durch die gleichen Produktionen modelliert werden und das Enthaltensein eines Musters in einem anderen nicht zur Konstruktion möglichst vieler identischer Produktionen führen muß, weil derartige Eigenschaften nicht überprüft werden (siehe hierzu auch Beispiel 19). Außerdem läßt die Fähigkeit des Inferenzverfahrens, aus wenigen Beispielen auf ein allgemeines Aufbauprinzip zu schließen und dies durch rekursive Produktionsfolgen zu verallgemeinern, noch sehr zu wünschen übrig.

Beispiel 20:

Das in Kapitel 3.2.1.2 geschilderte Grammatikkonstruktionsverfahren ist nicht einmal in der Lage, das äußerst einfache Aufbauprinzip der folgenden Stichprobe



zu erkennen und in Produktionen umzusetzen. Es liefert statt dessen eine Grammatik, die jedes Stichprobenelement durch eine Startproduktion und eine Kettenproduktion der Länge 2, 4 oder 6 erzeugt.

Die große Frage ist nun, was man im vorliegenden Fall eines konstruktiven, inkrementellen Inferenzverfahrens für Graphgrammatiken aus Ketten- und Sternproduktionen tun kann, um diesen wirklich gravierenden Mangel zu überwinden.

Eine Untersuchung der Mechanismen, mit denen andere Inferenzverfahren dieses Problem in den Griff zu bekommen versuchen, zeigt, daß dazu im wesentlichen die folgenden zwei Grundideen verfolgt werden:

- a) Die erste Möglichkeit zur Reduzierung der Produktionszahl in der Gesamtgrammatik besteht darin, die *Größe und Variabilität der rechten Seite von Produktionen stärker zu nor-*

mieren, wodurch sich die Wahrscheinlichkeit, daß in verschiedenen Einzelgrammatiken gleiche Produktionen auftreten, ganz zwangsläufig erhöht.

Beispiele hierfür sind in fast allen bekannten Inferenzverfahren enthalten, die Grammatiken in irgendeiner Art von Normalform generieren. So bestehen z.B. die rechten Seiten der meisten inferierten Zeichenkettengrammatiken aus maximal zwei Zeichen und die rechten Seiten der expansiven Baumgrammatiken nur aus Teilbäumen der Tiefe 0 oder 1.

- b) Die zweite Möglichkeit liegt in der *Ausnutzung von bisher gesammeltem Wissen über die Struktur der übrigen Stichprobenelemente* zur Konstruktion von möglichst wenigen zusätzlich notwendigen Produktionen.

Beispiele hierfür finden sich in Inferenzverfahren, die nach den am häufigsten vorkommenden gemeinsamen Teilstrukturen suchen oder vor Einführung einer neuen Produktion prüfen, ob deren Wirkung nicht auch durch die Anwendung bereits vorhandener Produktionen erzielt werden kann.

Die Umsetzung der ersten Grundidee zur Verbesserung des geschilderten Inferenzverfahrens ist Gegenstand der weiteren Erörterungen in diesem Kapitel, die Umsetzung der zweiten Grundidee behandelt das übernächste Kapitel 3.2.2.

Die Anwendung der ersten Idee zur Reduzierung der Produktionszahl in der Gesamtgrammatik macht eine Normierung der rechten Seite der Ketten- und Sternproduktionen auf einige wenige zulässige Teilstrukturen erforderlich. Die Art und Weise, wie dies zu geschehen hat, liegt bei der Form der Produktionen praktisch auf der Hand: nämlich durch *Beschränkung der Kettenlänge* bei Kettenproduktionen bzw. durch *Beschränkung des Sterngrades* bei Sternproduktionen. Hierdurch wird die Größe der Graphen auf der rechten Seite der Produktionen auf Ketten bis zu einer maximalen Länge m und Sterne bis zu einem maximalen Ausgangsgrad n eingeschränkt, wodurch sich die Wahrscheinlichkeit dafür, daß gleiche Graphstrukturen in verschiedenen Einzelgrammatiken durch gleiche Produktionen modelliert werden, bei geeigneter Wahl von m und n vergrößert.

Beispiel 21:

Für den Fall, daß die vom Inferenzverfahren gelieferten Grammatiken nur Kettenproduktionen der Länge ≤ 2 enthalten dürfen, könnte die Inferenz einer Grammatik für die Stichprobe aus Beispiel 20 folgendermaßen ablaufen:

- Das erste Stichprobenelement wird erzeugt durch die zur ersten Zwischengrammatik avancierende Einzelgrammatik

$$1. \quad \text{START} \quad \Rightarrow \quad \boxed{K}$$

$$2. \quad \boxed{K} \quad \Rightarrow \quad (K) \xrightarrow{A} (K) \xrightarrow{B} (K)$$

- Das zweite Stichprobenelement erweitert diese Grammatik um die Produktion

$$3. \quad \boxed{K} \quad \Rightarrow \quad (K) \xrightarrow{A} (K) \xrightarrow{B} \boxed{K}$$

- Das dritte und weitere Stichprobenelemente mit dem gleichen Konstruktionsprinzip ändern nichts mehr an der idealen Lösungsgrammatik bestehend aus den Produktionen 1, 2 und 3.

Leider sind Stichproben normalerweise nicht so einfach strukturiert wie die aus Beispiel 20, und es ist i.a. nicht auf den ersten Blick erkennbar, welche Parameter m und n für die maximale Kettenlänge und den maximalen Sterngrad die geeignetsten sind. So könnte z.B. die Stichprobe aus Beispiel 19 mit $m = 2$ wie in Beispiel 21 keine besonders gute, mit $m = 3$ hingegen eine ideale Grammatik liefern.

Experimente mit verschiedenen strukturierten Stichproben haben ergeben, daß es wenig sinnvoll ist, einen festen Wert für die Parameter m und n in das Inferenzverfahren einzubauen und daß auch alle Versuche, die Parameter aus den mittleren Knoten- und Kantenzahlen der Stichprobenelemente berechnen zu wollen, i.a. nicht zu optimalen Ergebnissen führen (siehe Beispiel 22 und [Du 79]).

Aus diesen Erfahrungen heraus wurde der Weg favorisiert, die Parameter zu Beginn des Inferenzprozesses vom Benutzer eingeben zu lassen. Damit hat man die Möglichkeit, verschiedene Parameter auszuprobieren und die inferierten Grammatiken miteinander zu vergleichen, ohne den beim systematischen Durchprobieren aller in Frage kommenden Parameter anfallenden Rechenaufwand investieren zu müssen. Meist ist es nämlich so, daß man sehr kleine und sehr große Parameter von vornherein außer Betracht lassen kann, weil diese die Stichprobe zu sehr bzw. zu wenig verallgemeinern. Das Finden der geeignetsten Parameter gleicht einem Balanceakt zwischen zu großer Allgemeinheit bzw. Spezialisierung der inferierten Grammatik und wird oft auch von der geplanten Anwendung beeinflusst. Wenn es z.B. wichtig ist, zwei relativ ähnlich strukturierte Klassen von Mustern gut zu trennen, empfiehlt sich die Wahl großer Parameter zur Inferenz einer die Stichprobe möglichst wenig verallgemeinernden Grammatik, während bei sehr verschieden strukturierten Musterklassen und bei Stichproben aus wenigen Elementen kleinere Parameter und eine die Stichprobe stärker verallgemeinernde Grammatik angebracht sind.

Nachdem auch die zuletzt aufgeführten Gesichtspunkte nur unzureichend automatisch erfaßbar und zu berücksichtigen sind, erscheint die Vorgehensweise, erst den Benutzer eine Vorauswahl geeigneter Parameter treffen zu lassen, dann das Inferenzprogramm Grammatiken und Kriterien zu ihrer Beurteilung berechnen zu lassen und die Endauswahl einer für die jeweilige Anwendung optimalen Grammatik wieder dem Benutzer zu übertragen, als die derzeit effizienteste und sinnvollste Arbeitsteilung beim Einsatz von Inferenzverfahren für verschiedene Aufgabenstellungen.

In Bezug auf die Generierung von Einzelgrammatiken bringt die Einführung der parametergesteuerten Grammatikkonstruktion kleine Veränderungen gegenüber der in Kapitel 3.2.1.2 geschilderten Vorgehensweise mit sich; alle anderen Phasen des Inferenzverfahrens bleiben davon unberührt.

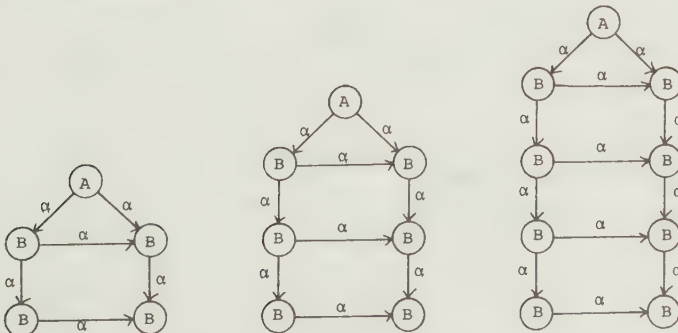
Die Algorithmen zur Generierung der Einzelgrammatiken für jeweils ein Stichprobenelement suchen im Graphen nur noch nach

gerichteten Kantenzügen bis zur Länge m und nach aus einem Knoten auslaufenden Kanten bis zum Ausgangsgrad n und lassen im Graphen vorhandene größere Ketten oder Sterne völlig außer Betracht. Dies bedeutet, daß als Startknoten der erste Knoten in der Knotenliste, der Ausgangsknoten einer Kette der Länge $\geq m$ ist, gewählt wird und dieser Knoten besonders bei kleinem m nicht mit dem Ausgangsknoten der längsten Bahn im Graphen identisch sein muß. Für den Fall, daß alle Bahnen im Graphen kürzer als m sind, ändert sich nichts gegenüber der bisherigen Vorgehensweise. Bei der Konstruktion der Kettenproduktionen wird die jeweils nächste gefundene Kette der Länge m unabhängig vom Vorhandensein noch längerer Ketten in eine Produktion umgesetzt. Bei der Generierung von Sternproduktionen führt dies dazu, daß für den Fall, daß ein Knoten mehr als n noch nicht erfaßte auslaufende Kanten hat, mehrere Sternproduktionen mit diesem Knoten als Sternausgangsknoten konstruiert werden müssen.

Diese Vorgehensweise ist effizienter als die Suche nach längsten Ketten, die man dann sowieso zerstückeln müßte, und bringt für Stichproben mit einem regelmäßigen Zuwachs in Verarbeitungsrichtung bessere Grammatiken als andere Strategien. Für den Fall, daß dadurch unerwünschte Zersplitterungseffekte auftreten, kann man diese durch die Wahl größerer Parameter vermindern bzw. aufheben.

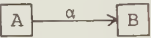
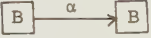
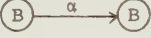
Beispiel 22:

Gegeben sei die folgende Stichprobe von "Häuschen mit beliebig vielen Stockwerken":

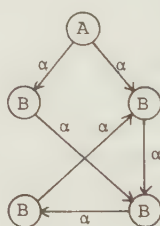
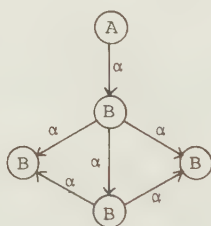
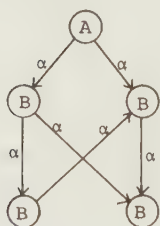


Aus der Struktur der Muster ist ersichtlich, daß für die maximale Kettenlänge Werte zwischen 1 und 5 und für den maximalen Sterngrad nur der Wert 1 in Frage kommen. Parameterwerte, die höher liegen als die genannten, führen zum gleichen Ergebnis wie $m = 5$ und $n = 1$, brauchen also nicht gesondert untersucht zu werden.

Für $m = 1$ und $n = 1$ ergibt sich folgende Grammatik G_{11} :

1. START $\Rightarrow$ 
2.  $\Rightarrow$ 
3.  $\Rightarrow$ 
4.  $\Rightarrow$ 
5.   $\Rightarrow$ 

Diese Grammatik ist zwar sehr kurz und auch in der Lage, Häuschen mit vier und mehr Stockwerken zu erzeugen, das Problem hierbei ist jedoch, daß viele Graphen wie z.B. die folgenden



in $L(G_1)$ enthalten sind, die mit den Häuschen der Stichprobe keine Ähnlichkeit mehr haben.

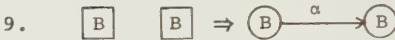
Für $m = 2$ und $n = 1$ erhält man G_{21} :

1. START $\Rightarrow$ $\boxed{A}$
2. $\boxed{A}$ $\Rightarrow$ $\boxed{A} \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B}$
3. $\boxed{B}$ $\Rightarrow$ $\bigcirc B \xrightarrow{\alpha} \bigcirc B \xrightarrow{\alpha} \boxed{B}$
4. $\boxed{B}$ $\Rightarrow$ $\bigcirc B \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B}$
5. $\boxed{A}$ $\boxed{B}$ $\Rightarrow$ $\bigcirc A \xrightarrow{\alpha} \boxed{B}$
6. $\boxed{B}$ $\boxed{B}$ $\Rightarrow$ $\bigcirc B \xrightarrow{\alpha} \bigcirc B$
7. $\boxed{B}$ $\boxed{B}$ $\Rightarrow$ $\bigcirc B \xrightarrow{\alpha} \boxed{B}$

Diese Grammatik enthält zwei Produktionen mehr und ist wie G_{11} imstande, Häuschen mit beliebig vielen Stockwerken zu erzeugen. Sie ist jedoch schon viel weniger allgemein als G_{11} und kann z.B. von den drei angegebenen Graphen aus $L(G_{11})$ nur mehr den ersten erzeugen.

Für $m = 3$ und $n = 1$ liefert das Inferenzverfahren die Grammatik G_{31} :

1. START $\Rightarrow$ $\boxed{A}$
2. $\boxed{A}$ $\Rightarrow$ $\boxed{A} \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B}$
3. $\boxed{B}$ $\Rightarrow$ $\bigcirc B \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \bigcirc B \xrightarrow{\alpha} \boxed{B}$
4. $\boxed{B}$ $\Rightarrow$ $\bigcirc B \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B}$
5. $\boxed{B}$ $\Rightarrow$ $\bigcirc B \xrightarrow{\alpha} \bigcirc B \xrightarrow{\alpha} \boxed{B}$
6. $\boxed{B}$ $\boxed{B}$ $\Rightarrow$ $\bigcirc B \xrightarrow{\alpha} \bigcirc B \xrightarrow{\alpha} \bigcirc B$
7. $\boxed{A}$ $\boxed{B}$ $\Rightarrow$ $\bigcirc A \xrightarrow{\alpha} \bigcirc B$



die mit zwei Produktionen mehr als G_{21} wiederum vier- und mehrstöckige Häuser, jedoch keinen der angegebenen drei Graphen aus $L(G_{11})$ mehr erzeugen kann.

Die Grammatiken G_{41} für $m = 4$ und G_{51} für $m = 5$ bestehen aus jeweils neun Produktionen und können keinen der angegebenen "falschen" Graphen aus $L(G_{11})$ generieren. Im Vergleich zu G_{31} haben sie aber auch die Fähigkeit verloren, vier- oder mehrstöckige Häuser erzeugen zu können.

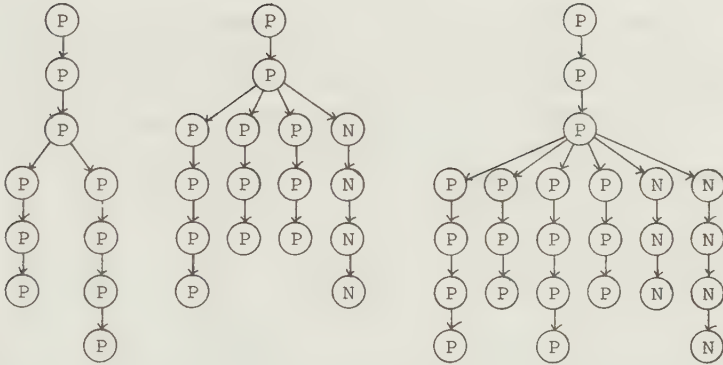
Dieses Beispiel demonstriert, wie die Parameterwahl den Umfang und die Verallgemeinerungsfähigkeit der inferierten Grammatiken beeinflussen kann. Im allgemeinen bewirkt eine Variation des Parameters m größere Veränderungen an der Grammatik als eine Variation von n . Die günstigsten Parameter liegen meist irgendwo in der Mitte des Bereichs der in Frage kommenden Parameter.

3.2.1.5 WEITERE BEISPIELE UND ERFAHRUNGEN

Die Rechenzeiten des in FORTRAN auf einer Rechenanlage vom Typ TR440-Dreifachprozessor realisierten Inferenzverfahrens betragen pro Stichprobenelement typischerweise fünf bis zehn Sekunden. Die Gesamtzeiten zur Inferenz einer Grammatik wachsen mit der Anzahl der verarbeiteten Stichprobenelemente annähernd linear an, weil sich der steigende Aufwand zur Vereinigung einer Einzelgrammatik mit der immer größer werdenden Zwischengrammatik nur wenig bemerkbar macht.

Beispiel 23:

Experimentiert wurde mit der Anwendung des geschilderten Inferenzverfahrens auf Graphdarstellungen der aus [FuBh 73] bekannten Bäume zur Beschreibung von Blasenkammerereignissen wie z.B.



mit Verzweigungsmöglichkeiten einer positiven Teilchenspur (P-Spur) in

- zwei P-Spuren oder
- drei P-Spuren und eine negative Teilchenspur (N-Spur) oder
- vier P-Spuren und eine N-Spur.

Die Verarbeitung von zehn verschiedenen Blasenkammerereignisgraphen mit verschiedenen Parametern liefert in dreißig bis vierzig Sekunden Rechenzeit Grammatiken mit jeweils einer Startproduktion und elf bis dreizehn Kettenproduktionen der Form a).

Alle inferierten Grammatiken können auch Graphen mit drei oder fünf Spuren beliebiger Krümmung erzeugen, weil das Inferenzverfahren nicht zählen kann, wie oft in einem nichtterminalen Knoten eine bestimmte Kette eingehängt werden darf. Ein Versuch, diese unerwünschte Eigenschaft durch Umbenennung der Verzweigungsknoten in P₂, P₄ und P₆ zu verändern, führt lediglich bei der Verzweigung in zwei P-Spuren zu der erhofften Verbesserung, daß aus einem P₂-Knoten immer genau zwei P-Spuren herausführen.

Falls sich die variable Länge der P-Spuren und N-Spuren in der Grammatik durch entsprechende rekursive Produktionen widerspiegeln soll, darf man den Parameter m nicht maximal wählen, sondern muß dafür sorgen, daß mindestens eine P-Spur und eine N-Spur durch mehr als nur eine Kettenproduktion aufgebaut werden.

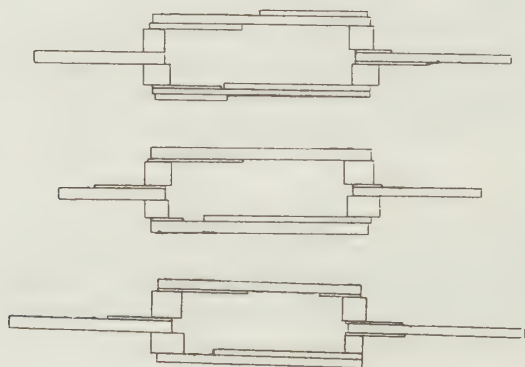
Zur Verarbeitung von Bäumen ist insgesamt festzustellen, daß das Graphgrammatikinferenzverfahren für Stichproben aus binären Bäumen ganz brauchbare Resultate liefert, wohingegen es bei nichtbinären Bäumen die Zahl der Nachfolger eines Knotens nicht stichprobengetreu erfaßt, sondern in fast beliebiger Weise verallgemeinert.

Beispiel 24:

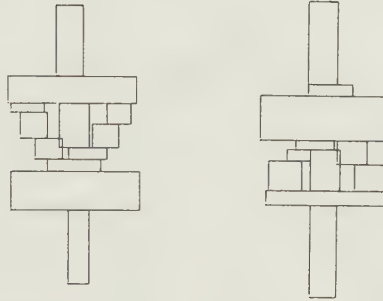
Die Anwendung des Inferenzverfahrens auf Stichproben von Graphen, die aus der Vorverarbeitung von realen Mustern gewonnen wurden, gestaltete sich problematisch, weil das am gleichen Lehrstuhl durchgeführte Projekt zur Verarbeitung von Stromlaufplänen [Bu 78, Bl 80] bedingt durch den langfristigen Ausfall des Bildaufnahmegerätes keine ausreichende Stichprobe von Bildgraphen zur Beschreibung von Schaltelementen [Bl 80] liefern konnte und es auch nicht möglich war, geeignetes Datenmaterial von anderer Stelle zur Verfügung gestellt zu bekommen.

Deswegen konnten nur relativ wenige aus der Vorverarbeitung von realen Schaltplänen gewonnene Bildgraphen für die Schaltelemente Widerstand und Diode als Stichprobengraphen für das Inferenzverfahren verwendet werden. Eine Vorstellung vom Zustandekommen dieser Bildgraphen sollen die folgenden Plotterzeichnungen vermitteln:

Widerstände:



Dioden:

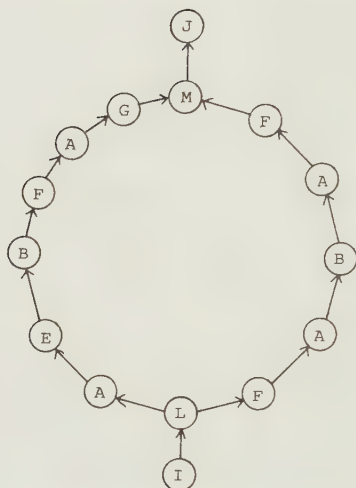


Die Knoten der Bildgraphen repräsentieren Gruppen von zusammenhängenden schwarzen Punkten, die zu sogenannten Primärkomponenten zusammengefaßt werden und in den Plotterzeichnungen durch ihr umschreibendes Rechteck dargestellt sind. Die Kanten der Bildgraphen entsprechen der Relation "Aneinandergrenzen von zwei Primärkomponenten" und beziehen ihre Orientierung aus der Verarbeitungsreihenfolge, d.h. der Quellknoten jeder Kante wurde früher als Primärkomponente erkannt als der Zielknoten.

Um eine für das Inferenzverfahren geeignete Knotenmarkierungsvorschrift zu gewinnen, wurde jeder Knoten mit einer die vier Merkmale Länge und Breite der Primärkomponente sowie positiver und negativer Grad des Knotens repräsentierenden Markierung versehen. Dies bedeutet, daß je zwei gleichmarkierte Knoten für zwei Primärkomponenten mit ungefähr gleicher Form und gleicher Anzahl ein- und auslaufender Kanten stehen und sich zwei verschieden markierte Knoten in mindestens einem der vier Merkmale signifikant unterscheiden.

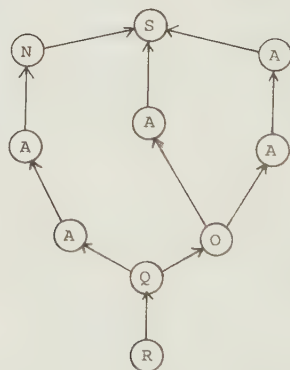
Die so erhaltenen Stichproben von Graphdarstellungen für Widerstände und Dioden wurden ergänzt durch weitere "simulierte" Graphen, die auf der Basis von naheliegenden Variationsmöglichkeiten in der Linienzeichnung wie z.B. leichte Schräglage eines Schaltelements oder unterschiedliche Liniendicke von Hand erzeugt wurden, so daß die beiden Stichproben zwanzig verschiedene Prototypen für Widerstandsgraphen wie z.B.

W_1 :



und zwanzig verschiedene Prototypen für Diodengraphen wie z.B.

D_1 :



umfassen.

Die mit den maximalen Parametern $m = 10$ und $n = 10$ inferierten Einzelgrammatiken für die Widerstände haben alle eine ähnliche Struktur wie die folgende Einzelgrammatik G_{W_1} für das Stichprobenelement W_1 :

1. START $\Rightarrow$ $\boxed{\text{I}}$

2. $\boxed{\text{I}}$ $\Rightarrow$ $(\text{I}) \rightarrow (\boxed{\text{L}}) \rightarrow (\text{A}) \rightarrow (\text{E}) \rightarrow (\text{B}) \rightarrow (\text{F}) \rightarrow (\text{A}) \rightarrow (\text{G}) \rightarrow (\boxed{\text{M}}) \rightarrow (\text{J})$

3. $\boxed{\text{L}}$ $\boxed{\text{M}}$ $\Rightarrow$ $(\text{L}) \rightarrow (\text{F}) \rightarrow (\text{A}) \rightarrow (\text{B}) \rightarrow (\text{A}) \rightarrow (\text{F}) \rightarrow (\text{M})$

Die Vereinigung der Einzelgrammatiken für die zwanzig Widerstandsgraphen ergibt nach fünfundsiebzig Sekunden Rechenzeit eine Gesamtgrammatik G_W , die aus drei Startproduktionen, zwanzig Kettenproduktionen der Form a) und zwanzig Kettenproduktionen der Form c) besteht. Jede Kettenproduktion modelliert dabei die Erzeugung einer rechten bzw. linken Hälfte eines Widerstandes.

Die in gleicher Weise mit den Parametern $m = 10$ und $n = 10$ inferierten Einzelgrammatiken für die Dioden weisen eine etwas größere strukturelle Vielfalt auf als die Einzelgrammatiken für die Widerstände. Die Einzelgrammatik G_{D1} für das Stichprobenelement D_1 hat z.B. folgendes Aussehen:

1. START $\Rightarrow$ $\boxed{\text{R}}$

2. $\boxed{\text{R}}$ $\Rightarrow$ $(\text{R}) \rightarrow (\boxed{\text{Q}}) \rightarrow (\boxed{\text{O}}) \rightarrow (\text{A}) \rightarrow (\text{A}) \rightarrow (\boxed{\text{S}})$

3. $\boxed{\text{Q}}$ $\boxed{\text{S}}$ $\Rightarrow$ $(\text{Q}) \rightarrow (\text{A}) \rightarrow (\text{A}) \rightarrow (\text{N}) \rightarrow (\boxed{\text{S}})$

4. $\boxed{\text{O}}$ $\boxed{\text{S}}$ $\Rightarrow$ $(\text{O}) \rightarrow (\text{A}) \rightarrow (\text{S})$

Die Gesamtgrammatik G_D für die zwanzig Diodengraphen erhält man nach nur fünfundfünfzig Sekunden Rechenzeit. G_D umfaßt insgesamt dreiunddreißig Produktionen, die sich in drei Startproduktionen, fünfzehn Kettenproduktionen der Form a) und fünfzehn Kettenproduktionen der Form c) unterteilen. Die geringere Anzahl von Produktionen im Vergleich zur Grammatik G_W ist darauf zurückzuführen, daß die drei Kettenproduktionen zum Aufbau einer Diode kürzer sind und somit weniger Variationsmöglichkeiten enthalten, wodurch sich bei der Vereinigung der

Einzelgrammatiken öfter eine Gelegenheit zur Zusammenfassung identischer Produktionen ergibt.

3.2.2 INFERENZ MIT GEDÄCHTNIS

Dieses Kapitel ist geprägt von dem Ziel, den Konstruktionsprozeß der Gesamtgrammatik zu stabilisieren und zu verbessern, indem man die Abhängigkeit von der Wahl der richtigen Parameter und der Reihenfolge des Auffindens gleichartiger Teilstrukturen reduziert. Der zur Erreichung dieses Ziels gewählte Ansatz besteht im wesentlichen darin, das Wissen über die Struktur der bisher verarbeiteten Stichprobenelemente bei der Konstruktion der Einzelgrammatiken für neue Stichprobenelemente möglichst geschickt auszunutzen.

Diese Vorgehensweise der Auswertung früherer Erfahrungen zur Lösung neuer Aufgabenstellungen ähnlicher Art orientiert sich an einem Grundprinzip menschlichen bzw. maschinellen Lernens, das sich in den verschiedensten Situationen anwenden läßt und bewährt hat. Demzufolge ist zu erwarten, daß die in diesem Zusammenhang auftretenden Probleme in erster Linie auf der Ebene der Integration dieses Prinzips in den vorhandenen Inferenzalgorithmus liegen werden.

Die inkrementelle Vorgehensweise bei der Grammatikkonstruktion läßt klar erkennen, daß alles Wissen über die Struktur der bisher verarbeiteten Stichprobenelemente in der aktuellen Zwischengrammatik enthalten sein muß und sich im Verlauf des Inferenzprozesses schrittweise erweitert und vervollständigt. Die aktuelle Zwischengrammatik stellt also eine Art "Gedächtnis" oder internes Modell für die bisher gesammelten Erfahrungen über die Struktur von Stichprobenelementen dar.

Die nächstliegende Frage ist nun, wie sich diese in der aktuellen Zwischengrammatik enthaltene Information nutzen läßt, um den Zuwachs an neuen Produktionen für die Gesamtgrammatik möglichst gering zu halten.

3.2.2.1 KONSTRUKTION VON EINZELGRAMMATIKEN UNTER VERWENDUNG DER AKTUELLEN ZWISCHENGRAMMATIK

Die zu diesem Zweck entwickelte Strategie sieht vor, daß die Einzelgrammatik für das erste Stichprobenelement genau wie bei der Inferenz ohne Gedächtnis aus Ketten- und Sternproduktionen kleiner oder gleich der Größe der eingegebenen Parameter m und n konstruiert wird. Diese erste Einzelgrammatik wird innerhalb der Ketten- und Sternproduktionen nach abnehmender Größe der erzeugten Teilgraphen sortiert, wobei unter Größe in diesem Fall die Zahl der durch die Produktion erzeugten Kanten, d.h. die Länge der Kette bzw. der Ausgangsgrad des Sterns, verstanden werden soll. Die erste aktuelle Zwischengrammatik des Inferenzprozesses ergibt sich aus der nach Größe sortierten ersten Einzelgrammatik.

Für jeden weiteren zu verarbeitenden Graphen wird nun abweichend von der Vorgehensweise des Inferenzverfahrens ohne Gedächtnis zuerst versucht, möglichst viele und möglichst große Teile des Graphen durch bereits in der Zwischengrammatik enthaltene Produktionen zu erzeugen und nur für die damit nicht generierbaren Teile des Graphen neue Produktionen gemäß der bisherigen Gesamtstrategie zu konstruieren.

Die Vorgehensweise im einzelnen soll durch das folgende Struktogramm [ScFl 79] veranschaulicht werden:

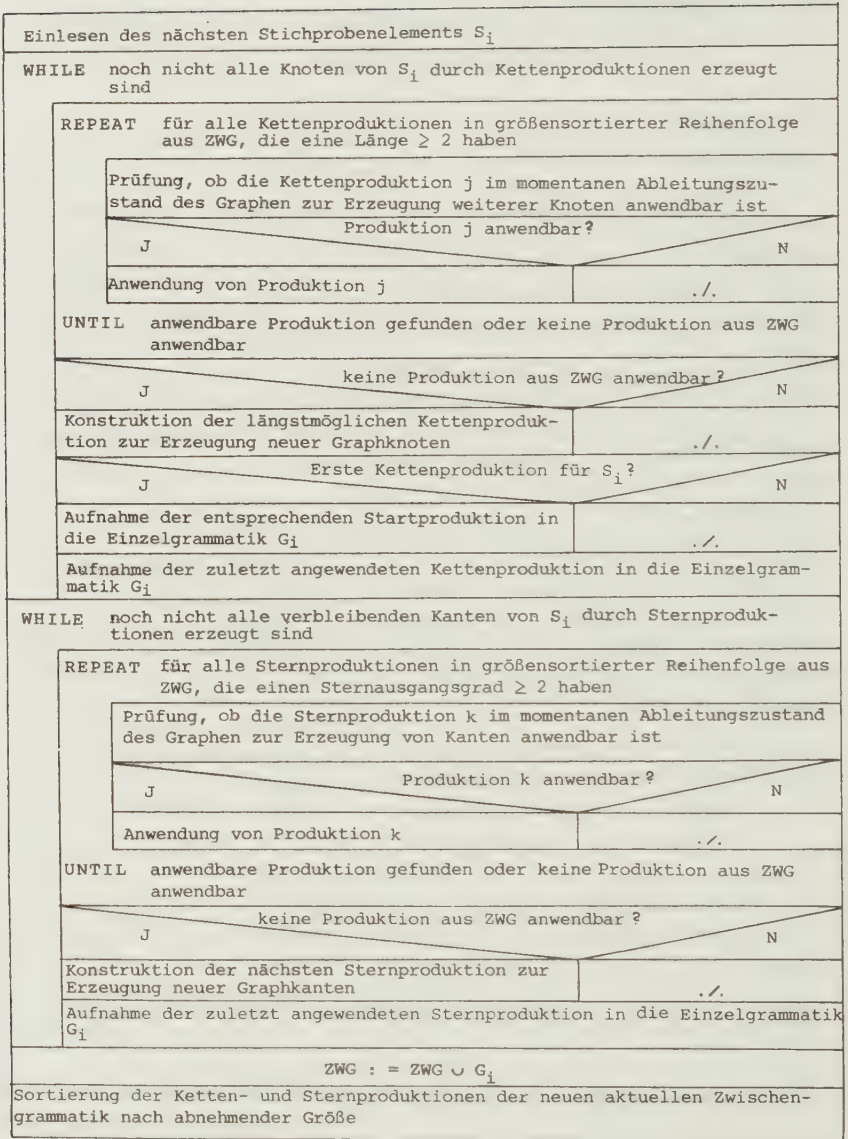


Bild 7 Struktogramm der Erweiterung der aktuellen Zwischengrammatik ZWG um eine neue Einzelgrammatik beim Inferenzalgorithmus mit Gedächtnis

Das Struktogramm von Bild 7 ersetzt beim Inferenzverfahren mit Gedächtnis die drei untersten Blöcke von Bild 6. Das Einlesen des nächsten Graphen aus der Stichprobe und die Konstruktion einer Startproduktion sind bei beiden Inferenzverfahren identisch. Die Vereinigung der aktuellen Zwischengrammatik ZWG mit einer Einzelgrammatik G_1 erfolgt auch auf dieselbe Weise wie beim Inferenzverfahren ohne Gedächtnis; es ist jedoch zweckmäßig, diesen Schritt beim Inferenzverfahren mit Gedächtnis mit einer Größensortierung der Produktionen zu verbinden, um dadurch die Suche nach der größtmöglichen anwendbaren Produktion zu erleichtern. Beim Inferenzverfahren ohne Gedächtnis hingegen ist es vollkommen unwichtig, ob, wann und nach welchen Kriterien die Produktionen der inferierten Grammatik sortiert werden, weshalb dieser Punkt auch in Bild 6 nicht enthalten ist.

Derjenige Teil des Struktogramms, der die Konstruktion der Ketten- und Sternproduktionen einer Einzelgrammatik G_1 beschreibt, teilt sich genau wie bei der Inferenz ohne Gedächtnis in einen Block zur Konstruktion der Kettenproduktionen und einen Block zur Konstruktion der Sternproduktionen auf. Das Inferenzverfahren mit Gedächtnis enthält zusätzlich die beiden repeat-until-Blöcke zur Suche nach einer möglichst großen anwendbaren Produktion aus ZWG und durchläuft nur dann, wenn diese Suche erfolglos war, das bisherige Verfahren zur Konstruktion der nächsten Ketten- bzw. Sternproduktion.

Die Prüfung auf Anwendbarkeit einer in ZWG enthaltenen Ketten- bzw. Sternproduktion soll gewährleisten, daß gleiche Substrukturen in verschiedenen Graphen nach Möglichkeit durch ein- und dieselbe Produktion modelliert werden. Nachdem aber die Reihenfolge der Abarbeitung der Knoten eines Graphen und die Umgebung der Knoten einer Substruktur einen nur schwer auszu-schaltenden Einfluß auf den Zeitpunkt der Terminalwerdung eines Graphknotens ausüben, hat es sich als vorteilhaft erwiesen, bei der Suche nach übereinstimmenden Teilstrukturen nur die Knoten- und Kantenmarkierungen zu vergleichen und die Überprüfung, ob auch die Eigenschaften terminal bzw. nichtterminal bei allen Knoten übereinstimmen, vorerst auszuklammern. Für den Fall, daß

sich hierbei später Unterschiede ergeben sollten, muß eine neue Produktion konstruiert werden, die sich von der in ZWG enthaltenen nur durch die Verteilung der terminalen Knoten auf der rechten Seite unterscheidet. Dies bedeutet dann zwar doch eine an sich unerwünschte Vergrößerung der Zwischengrammatik, die wegen der schnelleren Anwendbarkeitsüberprüfung von Produktionen mit strukturgleicher rechter Seite aber immer noch weniger Suchaufwand bei der Analyse verursacht als die gleiche Anzahl von Produktionen mit strukturungleicher rechter Seite.

Die Prüfung auf Anwendbarkeit einer in ZWG enthaltenen Ketten- oder Sternproduktion läuft darauf hinaus, zuerst das Vorhandensein der linken Seite X_1 im momentanen Zustand der Graph-Satzform $X_i \not\subseteq S_i$ und danach das Vorhandensein der einzusetzenden rechten Seite X_r an der entsprechenden Position im Graphen S_i festzustellen. Die Suche nach den nichtterminalen Knoten der linken Seite einer Produktion erfordert im Falle einer Kettenproduktion mit nur einem Knoten auf der linken Seite die Untersuchung aller einzelnen Graphknoten, im Falle einer Kettenproduktion mit zwei Knoten auf der linken Seite oder einer Sternproduktion vom Typ S_1 die Untersuchung aller Paare von Graphknoten und im Falle einer Sternproduktion vom Typ S_n die Untersuchung aller $(n + 1)$ -Tupel von Knoten in X_i in bezug auf ihre Verwendbarkeit als linke Seite der Produktion. Hieraus ist leicht ablesbar, daß der Suchaufwand für eine passende linke Seite in erster Linie mit der Zahl der darin enthaltenen Kontextknoten ziemlich rasch ansteigt. Hat man nun auf diese Weise eine oder mehrere mögliche linke Seiten gefunden, so wird überprüft, ob die Ersetzung der linken Seite durch die rechte aus X_i einen weiteren Teilgraphen $X_{i+1} \subseteq S_i$ entstehen läßt, d.h. ob alle durch die Anwendung der Produktion hinzukommenden Knoten und Kanten in S_i , aber noch nicht in X_i enthalten sind. Ist dies der Fall, so wird auch ohne völlige Übereinstimmung der Verteilung der terminalen Knoten angenommen, daß es sich hier um gleiche Substrukturen handelt und die Produktion wird mit der für S_i passenden Verteilung der terminalen bzw. nichtterminalen Knoten angewendet und in die Einzel-

grammatik G_i aufgenommen.

Die Einschränkung, nur Ketten- oder Sternproduktionen ≥ 2 auf ihre Anwendbarkeit hin zu untersuchen, soll der Tendenz zur Verwendung kleinerer Produktionen, die beim Inferenzverfahren mit Gedächtnis in stärkerem Maße als beim Inferenzverfahren ohne Gedächtnis zu beobachten ist, etwas entgegenwirken; zumal in allen Fällen, in denen eine Produktion der Größe 1 zur Vervollständigung einer Teilstruktur benötigt wird, beide Vorgehensweisen dieselbe Produktion generieren, nur braucht das Inferenzverfahren mit Gedächtnis entsprechend länger, bis es die evtl. in ZWG vorhandene anwendbare Produktion gefunden hat. Für Anwendungsfälle, in denen hauptsächlich größere Teilstrukturen vorkommen, ist es auch möglich, die Schwelle für die Größe der in ZWG zu überprüfenden Produktionen noch höher anzusetzen.

Beispiel 25:

Gegeben sei erneut die Häuschen-Stichprobe aus Beispiel 22. Das Inferenzverfahren mit Gedächtnis liefert nach Eingabe der eigentlich zu groß gewählten Parameter $m = 10$ und $n = 10$ für das erste Stichprobenelement die Grammatik G_1 :

1. START $\Rightarrow$ A

2. A $\Rightarrow$ A $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B

3. B B $\Rightarrow$ B $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B

4. A B $\Rightarrow$ A $\xrightarrow{\alpha}$ B

Diese Grammatik bildet die Grundlage für die Konstruktion der Einzelgrammatik G_2 für das zweite Stichprobenelement, die folgendermaßen abläuft:

Die Überprüfung der Anwendbarkeit der Kettenproduktion Nr.2 aus G_1 liefert ein positives Resultat und führt zur Aufnahme der Startproduktion

$$1. \quad \text{START} \quad \Rightarrow \quad \boxed{A}$$

und der Kettenproduktion

$$2. \quad \boxed{A} \quad \Rightarrow \quad \boxed{A} \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B}$$

in die Grammatik G_2 .

In dem durch Anwendung der Produktionen 1 und 2 entstehenden Teilgraphen X_2 ist die Kettenproduktion Nr.2 kein zweites Mal anwendbar, weil dadurch eine in S_2 nicht enthaltene Teilstruktur entstehen würde, hingegen ist die Produktion 3 aus G_1 anwendbar, wenn man von der fehlenden Übereinstimmung des terminalen Zustands des zweiten und dritten Knotens der Kette absieht. Dies führt zur Aufnahme der folgenden Produktion in die Grammatik G_2 :

$$3. \quad \boxed{B} \quad \boxed{B} \quad \Rightarrow \quad (\textcircled{B}) \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B}$$

Zur Konstruktion des nun noch fehlenden "Stockwerks" ist keine der Kettenproduktionen aus G_1 verwendbar, deshalb wird als letzte Kettenproduktion neu hinzugenommen:

$$4. \quad \boxed{B} \quad \Rightarrow \quad (\textcircled{B}) \xrightarrow{\alpha} (\textcircled{B}) \xrightarrow{\alpha} \boxed{B}$$

Die beiden zur Vervollständigung noch fehlenden Sternproduktionen werden neu konstruiert, da G_1 nur Sternproduktionen vom Grad < 2 enthält:

$$5. \quad \boxed{B} \quad \boxed{B} \quad \Rightarrow \quad (\textcircled{A}) \xrightarrow{\alpha} (\textcircled{B})$$

$$6. \quad \boxed{B} \quad \boxed{B} \quad \Rightarrow \quad (\textcircled{B}) \xrightarrow{\alpha} (\textcircled{B})$$

Die neue aktuelle Zwischengrammatik aus $G_1 \cup G_2$ hat nun folgende Gestalt:

1. START $\Rightarrow$ A
2. A $\Rightarrow$ A $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B
3. B B $\Rightarrow$ B $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B
4. B B $\Rightarrow$ B $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B
5. B $\Rightarrow$ B $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B
6. A B $\Rightarrow$ A $\xrightarrow{\alpha}$ B
7. B B $\Rightarrow$ B $\xrightarrow{\alpha}$ B

Die Grammatik G_3 für das dritte Stichprobenelement wird aufgebaut durch die Startproduktion

1. START $\Rightarrow$ A

und die Kettenproduktionen

2. A $\Rightarrow$ A $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B
3. B B $\Rightarrow$ B $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B
4. B $\Rightarrow$ B $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B
5. B $\Rightarrow$ B $\xrightarrow{\alpha}$ B $\xrightarrow{\alpha}$ B

die alle aus der Anwendbarkeit vorhandener Produktionen entstanden sind und von denen nur Produktion 4 bisher noch nicht in ZWG enthalten war. Die Sternproduktionen haben die Gestalt:

6. A B $\Rightarrow$ A $\xrightarrow{\alpha}$ B
7. B B $\Rightarrow$ B $\xrightarrow{\alpha}$ B
8. B B $\Rightarrow$ B $\xrightarrow{\alpha}$ B

Als größensortierte Endgrammatik G_{1010} für die Häuschen-Stichprobe erhält man:

1. START $\Rightarrow$ $\boxed{A}$
2. $\boxed{A}$ $\Rightarrow$ $\boxed{A} \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B}$
3. $\boxed{B}$ $\boxed{B}$ $\Rightarrow$ $\boxed{B} \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B}$
4. $\boxed{B}$ $\Rightarrow$ $\boxed{B} \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B}$
5. $\boxed{B}$ $\boxed{B}$ $\Rightarrow$ $\boxed{B} \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B}$
6. $\boxed{B}$ $\Rightarrow$ $\boxed{B} \xrightarrow{\alpha} \boxed{B} \xrightarrow{\alpha} \boxed{B}$
7. $\boxed{A}$ $\boxed{B}$ $\Rightarrow$ $\boxed{A} \xrightarrow{\alpha} \boxed{B}$
8. $\boxed{B}$ $\boxed{B}$ $\Rightarrow$ $\boxed{B} \xrightarrow{\alpha} \boxed{B}$
9. $\boxed{B}$ $\boxed{B}$ $\Rightarrow$ $\boxed{B} \xrightarrow{\alpha} \boxed{B}$

Diese Grammatik ist mit keiner der in Beispiel 22 inferierten Grammatiken identisch, hat aber große Ähnlichkeit mit G_{21} und G_{31} aus diesem Beispiel und ist in der Lage, weitere Häuschen mit beliebig vielen Stockwerken zu generieren.

Es zeigt sich an diesem Beispiel, daß das Inferenzverfahren mit Gedächtnis von der Wahl der "richtigen" Parameter weitgehend unabhängig arbeitet, aber dazu neigt, umfangreichere Grammatiken mit kleineren Produktionen zu liefern, wenn die zuerst eingegebenen Stichprobenelemente klein sind. Andererseits liefert eine nicht nach zunehmender Größe sortierte Verarbeitungsreihenfolge der Stichprobenelemente keine gut verallgemeinerungsfähigen Grammatiken mehr, sondern Ergebnisse ähnlich denen aus dem Inferenzverfahren ohne Gedächtnis bei zu groß gewählten Parametern.

3.2.2.2 WEITERE BEISPIELE UND ERFAHRUNGEN

Der Rechenzeitbedarf des Inferenzverfahrens mit Gedächtnis liegt schon allein aufgrund der beschriebenen Vorgehensweise bei der Konstruktion der Einzelgrammatiken in jedem Fall höher als der des Inferenzverfahrens ohne Gedächtnis. Der Aufwand für die Aufnahme eines einzelnen Stichprobenelements in die aktuelle Zwischengrammatik ist umso größer, je mehr Produktionen aus ZWG auf ihre Anwendbarkeit hin überprüft werden müssen und kann die Zeit für die Konstruktion einer Einzelgrammatik auf ein Mehrfaches der beim Inferenzverfahren ohne Gedächtnis benötigten Zeit anwachsen lassen. Die für gleiche Stichproben mit gleichen Parametern ermittelten Rechenzeiten der Inferenzverfahren mit bzw. ohne Gedächtnis differieren um circa zwanzig bis einhundert Prozent mit einer Häufung bei Werten um fünfzig Prozent.

Für diesen Mehraufwand erhält man Grammatiken, die typischerweise weniger und kleinere Produktionen als die vom Inferenzverfahren ohne Gedächtnis mit gleichen Parametern gelieferten Grammatiken enthalten, wobei der Unterschied zwischen beiden Grammatiken umso größer ausfällt, je mehr gemeinsame Teilstrukturen in den Graphen der Stichprobe zu finden sind. Meistens können die durch das Verfahren mit Gedächtnis inferierten Grammatiken auch eine größere Anzahl von terminalen Graphen erzeugen als die durch das Verfahren ohne Gedächtnis inferierten, d.h. sie verallgemeinern die Stichprobe stärker.

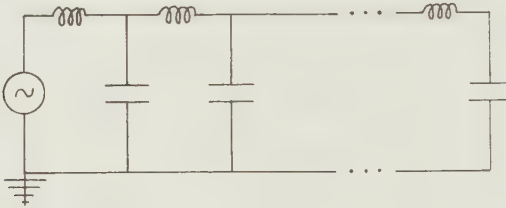
Zusammenfassend läßt sich sagen, daß das Inferenzverfahren mit Gedächtnis nicht in jedem Fall geeignetere Grammatiken liefert als das Inferenzverfahren ohne Gedächtnis, es ist aber aufgrund seiner weitgehenden Unabhängigkeit von der Wahl der richtigen Parameter unkomplizierter anzuwenden und empfiehlt sich besonders für Stichproben von Mustern, die viele wiederholt vorkommende Teilstrukturen enthalten.

Die größten Unterschiede zum Inferenzverfahren ohne Gedächtnis zeigen sich bei der Fähigkeit zur Erfassung von Rekursionen,

wie das folgende Beispiel noch einmal demonstrieren soll:

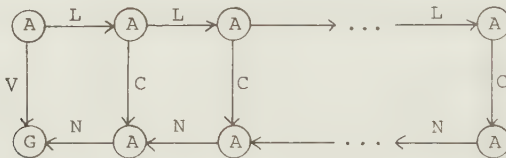
Beispiel 26:

Ausgangspunkt sei eine Menge von LC-Netzwerken der folgenden Gestalt



wie sie in [Fu 74] durch eine Baumgrammatik erzeugt werden.

Die Darstellung dieser Netzwerke durch Graphen der Form



wird als Stichprobe dem Inferenzverfahren mit Gedächtnis bei Wahl der Parameter $m \geq 3$ und $n \geq 1$ eingegeben. Schon nach Verarbeitung der drei Graphen mit einem, zwei und drei LC-Teilen entsteht nach neun Sekunden Rechenzeit folgende Grammatik:

1. START $\Rightarrow$ A
2. A A $\Rightarrow$ A $\xrightarrow{L}$ A $\xrightarrow{C}$ A $\xrightarrow{N}$ A
3. A $\Rightarrow$ A $\xrightarrow{L}$ A $\xrightarrow{C}$ A $\xrightarrow{N}$ G
4. A A $\Rightarrow$ A $\xrightarrow{L}$ A $\xrightarrow{C}$ A $\xrightarrow{N}$ A
5. A $\Rightarrow$ A $\xrightarrow{L}$ A $\xrightarrow{C}$ A $\xrightarrow{N}$ G
6. A G $\Rightarrow$ A $\xrightarrow{V}$ G

die das Hinzukommen eines LC-Teils durch die Produktion 2 modelliert und ihre Gestalt auch bei Eingabe von weiteren Stichprobenelementen mit mehr als drei LC-Teilen nicht mehr verändert.

4. SYNTAXANALYSE DER GRAPHGRAMMATIKEN

Das folgende Kapitel beschreibt ein nichtdeterministisches Syntaxanalyseverfahren für Graphgrammatiken der in Kapitel 3.1.2 spezifizierten Form und gibt Beispiele für seine Anwendung zur Klassifikation von durch Graphen dargestellten Mustern.

4.1 ALLGEMEINE VORBEMERKUNGEN

Arbeiten über neue Inferenzverfahren für Zeichenketten- oder Baumgrammatiken enden gewöhnlich mit einem Hinweis auf ein bekanntes Syntaxanalyseverfahren, das für die inferierte Klasse von Grammatiken angewendet werden kann. Leider ist dieser Weg im vorliegenden Falle nicht beschreitbar, weil einerseits noch so wenig Erfahrung mit der Syntaxanalyse von Graphgrammatiken existiert, daß es kein allgemein eingeführtes Verfahren für diese Aufgabenstellung gibt, und andererseits alle in der Literatur geschilderten Verfahren [My 72, Pf 72, Mi 75, Fr 76, De 78] irgendwelche Einschränkungen bezüglich der Klasse der behandelten Graphen bzw. Graphgrammatiken machen, die durch die in Kapitel 3.1.1 spezifizierten Stichprobengraphen und die in Kapitel 3.1.2 definierten Graphgrammatiken nicht eingehalten werden.

Vor diesem Hintergrund wird verständlich, warum ein Inferenzverfahren für Graphgrammatiken ohne dazugehörigen Syntaxanalysealgorithmus einem Auto ohne Räder gleicht, das zwar schön anzusehen, aber nicht vernünftig zu gebrauchen ist. Deshalb ist es in der bestehenden Situation unabdingbar, die Inferenzalgorithmen durch die Entwicklung eines eigenen Syntaxanalyseverfahrens zu ergänzen.

Dieses Syntaxanalyseverfahren darf nicht wie z.B. ein Compiler auf eine ganz bestimmte Grammatik hin konstruiert sein, sondern muß *beliebige Grammatiken* des in Kapitel 3.1.2 beschriebenen Typs vom Inferenzverfahren übernehmen und für beliebige Graphen entscheiden können, ob sie durch eine bestimmte Grammatik er-

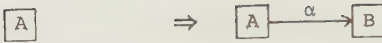
zeugbar sind oder nicht. Man kann sich das Ganze als eine Art universeller erkennender Automat für alle durch inferierte Graphgrammatiken modellierten Graphsprachen vorstellen.

Nachdem die gegebenen Graphgrammatiken nicht kontextfrei sind und auch keine erfolgversprechenden Vorrangrelationen ähnlich [Fr 76] auffindbar waren, kann man aufgrund der Erfahrungen im Zeichenkettenfall mit Sicherheit davon ausgehen, daß die Syntaxanalyse der ketten- und sternstrukturierten Graphgrammatiken nur *nichtdeterministisch* möglich sein wird. Dies hat zur Folge, daß das Verfahren die *Fähigkeit* besitzen muß, *Sackgassen* im Ableitungsprozeß zu erkennen und *rückgängig zu machen*.

Die einzige Basis, auf der man die Entwicklung des Verfahrens nun noch aufbauen kann, ist die Entscheidbarkeit des Graphproblems für monotone Graphsprachen (Satz 3b) aufgrund der endlichen Zahl von Möglichkeiten, einen bestimmten Graphen in einer Grammatik abzuleiten. Außerdem besteht die Hoffnung, vom Wissen über das Zustandekommen der Grammatik beim Inferenzprozeß für den Analysevorgang soviel profitieren zu können, daß man die ganz erhebliche Berechnungskomplexität des allgemeinen Graphproblems [Br 77, De 78] in eine für Anwendungen tragbare Größenordnung herunterdrücken kann.

Die prinzipielle Vorgehensweise des Analyseverfahrens besteht demzufolge hauptsächlich darin, die Menge aller Möglichkeiten, einen Graphen X mit einer monotonen Graphgrammatik G abzuleiten, soweit zu untersuchen, bis der Erfolg oder Mißerfolg dieses Vorhabens feststeht.

Die Frage, ob dies besser top-down oder bottom-up geschehen sollte, ist ohne eingehendere Beschäftigung mit den inferierten Grammatiken und der Strategie zur Suche nach einem Ansatz für die nächste Produktions- bzw. Reduktionsregel schwierig zu entscheiden. Eine Untersuchung der Grammatiken auf Eigenschaften, die bei Zeichenkettengrammatiken für die Analyserichtung von Bedeutung sind, ergibt, daß die Graphgrammatiken sowohl links-rekursive Produktionen wie



als auch rechtsrekursive Produktionen wie



und zudem eine Art beidseitig rekursiver Produktionen wie



enthalten dürfen. Rekursive Produktionen können bei einer Top-down-Strategie dann problematisch werden, wenn aus dem zur Verfügung stehenden Kontext nicht ersichtlich ist, wie oft sie hintereinander anwendbar sind, wie dies z.B. bei der Analyse von kontextfreien Zeichenkettensprachen mit Hilfe eines Kellerautomaten der Fall sein kann. Glücklicherweise sind Probleme dieser Art beim vorliegenden Graphanalyseproblem nicht zu erwarten, weil durch Inspektion der noch nicht erzeugten Teile des Graphen immer sofort festgestellt werden kann, ob eine bestimmte rekursive Produktion ein weiteres Mal anwendbar ist oder nicht.

Die inferierten Graphgrammatiken dürfen auch Paare von nicht eindeutig reduzierbaren Produktionen wie z.B.



enthalten, die bei einer Bottom-up-Strategie zu Situationen führen, in denen aus dem Kontext keine Anhaltspunkte für die richtige Entscheidung gewonnen werden können und sich erst zu einem späteren Zeitpunkt herausstellt, ob die gewählte Produktion richtig war oder wieder rückgängig gemacht werden muß. Solche Situationen sind unerwünscht, weil sie einen erheblichen Aufwand für das immer wieder notwendig werdende Zurücksetzen des Verfahrens auf einen Zustand vor der inzwischen als falsch

erkannten Entscheidung verursachen und deshalb nach Möglichkeit vermieden bzw. auf ein Minimum reduziert werden sollten. Zwar werden bei einer Top-down-Strategie durch Produktionen mit gleichen linken und verschiedenen rechten Seiten in gewisser Weise ähnliche Situationen hervorgerufen, doch hat man hier durch die Suche nach dem Vorhandensein der unterschiedlichen rechten Seiten im zu erzeugenden Graphen vielfach die Möglichkeit, eine nicht anwendbare Produktion gleich und nicht erst nach dem Umweg in eine Sackgasse auszusortieren.

4.2 SYNTAXANALYSEALGORITHMUS

Aus den zuletzt gemachten Vorbemerkungen ist ablesbar, daß im vorliegenden Fall ein Top-down-Verfahren die wohl geeignetere Vorgehensweise darstellt und deshalb wurde auch der realisierte Syntaxanalysealgorithmus als *Top-down-Verfahren* konzipiert.

Eine weitere wichtige strategische Entscheidung, die durch die Top-down-Vorgehensweise begünstigt wird, besteht in der Tatsache, daß bei der Suche nach der nächsten anwendbaren Produktion der Versuch gemacht wird, die *Strategie des Inferenzverfahrens nachzuahmen* und die Produktionen in der gleichen Reihenfolge anzuwenden, in der das Inferenzverfahren sie generiert haben würde. Dies bedeutet, daß sich die Analyse eines Graphen genau wie die Inferenz einer Einzelgrammatik im wesentlichen in zwei Etappen vollzieht:

- zunächst werden alle Knoten des Graphen mit den dazwischenliegenden Kanten durch Kettenproduktionen aufgebaut und
- dann werden alle noch fehlenden Kanten durch Sternproduktionen hinzugefügt.

Die weiteren Einzelheiten des Syntaxanalyseverfahrens sind dem folgenden Struktogramm zu entnehmen:

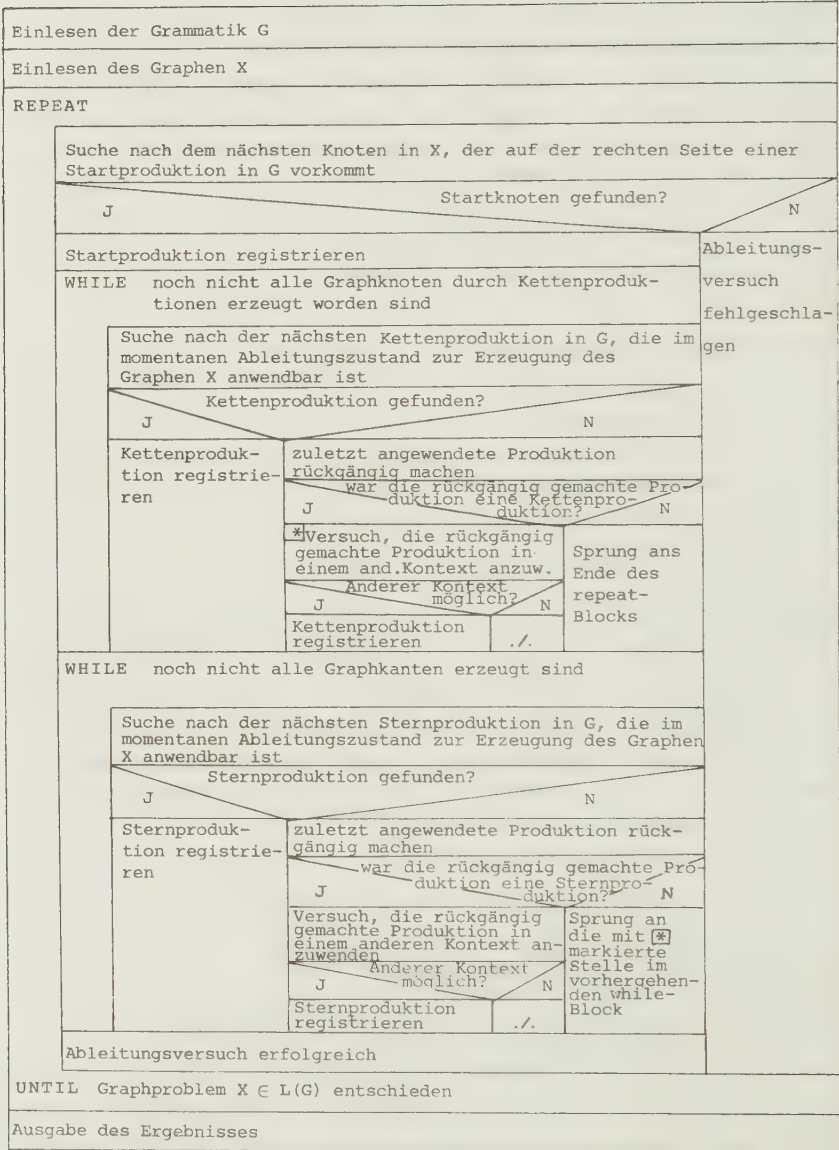


Bild 8 Struktogramm der Syntaxanalyse eines Graphen X mit
Hilfe einer Grammatik G

Aus dem Struktogramm von Bild 8 ist erkennbar, daß sich eine Sackgasse im Ableitungsprozeß darin äußert, daß zu einer Graph-Satzform, die noch nichtterminale Knoten enthält, keine zur Erzeugung des Graphen X anwendbare Folgeproduktion mehr gefunden wird. In dieser Situation muß die letzte Produktion rückgängig gemacht und ein anderer Ableitungsweg eingeschlagen werden. Dieser andere Ableitungsweg kann bestehen in

- a) der Anwendung derselben Produktion in einem anderen Kontext oder
- b) der Anwendung einer anderen Produktion.

Falls keine der beiden Möglichkeiten a) oder b) zu einem positiven Ergebnis führt, muß die Produktion davor rückgängig gemacht werden usw. bis man im ungünstigsten Fall die gesamte Ableitung von rückwärts aufgerollt hat und mit einem neuen Startknoten von vorne beginnen muß. Wenn alle in Frage kommenden Startknoten erfolglos ausprobiert wurden, gilt der Ableitungsversuch als fehlgeschlagen. Sobald eine Folge von Produktionen genau den Graphen X erzeugt, kann die Syntaxanalyse erfolgreich beendet werden.

Bezüglich der Anwendbarkeit einer Produktion gilt ähnliches wie in Kapitel 3.2.2, nur muß hier im Unterschied zu Kapitel 3.2.2 auch die Verteilung der nichtterminalen und terminalen Knoten auf der rechten Seite der Graphproduktion mit dem momentanen Zustand der Graph-Satzform exakt übereinstimmen.

Die Registrierung einer anwendbaren Produktion verlangt wegen der back-up-Möglichkeit nicht nur eine Speicherung der Produktionsnummer, sondern auch der Information darüber, welche Knoten und Kanten in der Graph-Satzform welchen Knoten und Kanten auf der rechten Seite der Produktion entsprechen, weil sonst der Zustand der Graph-Satzform vor Anwendung der Produktion unter Umständen nicht mehr eindeutig rekonstruierbar wäre.

Beispiel 27:

Gegeben sei die Grammatik G_{21} von Beispiel 22. Die Syntaxanalyse der Stichprobengraphen aus Beispiel 22 liefert für das erste Stichprobenelement die Produktionsfolge 1 2 3 5 6, für das zwei-

te 1 2 4 3 5 7 6 und für das dritte 1 2 4 4 3 5 7 7 6. Die gefundenen Ableitungen entsprechen genau der Vorgehensweise beim Inferenzalgorithmus.

Weitere analog zur Stichprobe aufgebaute "Häuschen" mit vier, fünf bzw. k Stockwerken sind erzeugbar durch die Produktionsfolgen 1 2 4³ 3 5 7³ 6, 1 2 4⁴ 3 5 7⁴ 6 bzw. 1 2 4 ^{$k-1$} 3 5 7 ^{$k-1$} 6.

Von den in Beispiel 22 angegebenen negativen Stichprobenelementen ist durch G_{21} nur das erste generierbar, und zwar durch die Produktionsfolge 1 2 3 5 6. Hieraus ist ersichtlich, daß durch die gleiche Produktionsfolge verschiedene terminale Graphen ableitbar sein können.

Auch ein Beispiel dafür, daß ein Graph durch verschiedene Produktionsfolgen erzeugbar sein kann, ist in der Grammatik G_{21} enthalten:

Das zweite Stichprobenelement von Beispiel 22 ist sowohl durch die Produktionsfolge 1 2 4 3 5 7 6 als auch durch die Produktionsfolge 1 2 4 4 5 6 6 ableitbar.

Der Rechenaufwand für die Syntaxanalyse eines Graphen ist in erster Linie abhängig von der Anzahl und Länge der Sackgassen, die das Verfahren durchlaufen muß, bis das Graphproblem entschieden ist. Die Möglichkeiten, in eine Sackgasse zu geraten, wachsen mit der Anzahl der in der Grammatik enthaltenen Produktionen und mit der Länge der Ableitung, die i.a. mit der Größe des Graphen zusammenhängt, sehr steil an und sind durch die bisher getroffenen Maßnahmen zur Aufwandsreduzierung wie z.B. der Trennung in nur in dieser Reihenfolge anwendbare Start-, Ketten- und Sternproduktionen noch nicht befriedigend einzudämmen.

Deshalb wurde untersucht, inwieweit die Reihenfolge, in der die Ketten- und Sternproduktionen auf ihre Anwendbarkeit hin überprüft werden, einen Einfluß auf die Rechenzeiten des Syntaxanalysealgorithmus ausübt und welche naheliegenden Ordnungskriterien besonders gute Ergebnisse liefern.

Ausgangspunkt der Untersuchungen war eine Ordnung der Produktionen getrennt nach Ketten- und Sternproduktionen in der

Reihenfolge, in der das Inferenzverfahren sie zeitlich aufeinanderfolgend konstruiert hatte. Eine Umordnung der Produktionen nach fallender Häufigkeit ihrer Anwendung zur Erzeugung der Stichprobe, wie sie z.B. bei eindeutigen und konsistenten kontextfreien Grammatiken zu guten Ergebnissen führt [LeFu 72], brachte keine signifikante Verbesserung der Analysezeiten mit sich. Die Gründe dafür sind zum einen, daß die ursprüngliche Ordnung der Produktionen in der Reihenfolge ihrer Konstruktion durch das Inferenzverfahren schon besser ist als eine vollkommen zufällig zustandgekommene Reihenfolge, und zum anderen, daß man auch hier die Beobachtung macht, daß kleine Produktionen i.a. häufiger angewendet werden als große. Dies hat zur Folge, daß die Ordnung nach fallender Anwendungshäufigkeit einer Ordnung nach zunehmender Größe der rechten Seiten sehr ähnlich ist und wegen der vielseitigeren Anwendbarkeit kleiner Produktionen besonders häufig in eine Sackgasse führt.

Das beste Mittel, diese Tendenz umzukehren, ist eine *Umordnung* der Produktionen *nach fallender Größe und bei Ketten- bzw. Sternproduktionen gleicher Größe nach fallender Anwendungshäufigkeit*. Durch diese Reihenfolge wird bewirkt, daß große Produktionen zwar oft vergeblich auf ihre Anwendbarkeit hin überprüft werden, aber in den meisten Fällen, in denen sie anwendbar sind, auch die richtige Folgeproduktion darstellen und nicht in eine Sackgasse führen. Das zuletzt geschilderte Ordnungskriterium für die Produktionen benötigt erkennbar geringere Rechenzeiten für die Analyse von durch die Grammatik erzeugbaren Graphen und wird deshalb bei allen Beispielen in dieser Arbeit verwendet. Das Ausmaß der hierdurch erreichbaren Zeitersparnis ist stark von der speziellen Gestalt der Grammatik abhängig und liegt für Grammatiken, bei denen das Ordnungskriterium zu einer Änderung der ursprünglichen Reihenfolge führt, in der Gegend von zehn bis dreißig Prozent.

4.3 WEITERE BEISPIELE UND ERFAHRUNGEN

Die Rechenzeiten für die Syntaxanalyse eines Graphen liegen im

allgemeinen im Bereich von einigen Sekunden und sind für Graphen, die durch die zugrundeliegende Grammatik erzeugbar sind, meist sogar niedriger als die Rechenzeiten zur Aufnahme desselben Graphen in die Grammatik, wohingegen sie für Graphen, die durch die zugrundeliegende Grammatik nicht erzeugbar sind, das zwei- bis zehnfache dieser Werte betragen können.

Der Aufwand für die Programmierung des Syntaxanalysealgorithmus liegt mit circa 800 FORTRAN-Befehlen etwas niedriger als der Aufwand für die Realisierung des Inferenzverfahrens ohne Gedächtnis (ca. 900 Befehle) bzw. des Inferenzverfahrens mit Gedächtnis (ca. 1200 Befehle).

Beispiel 28:

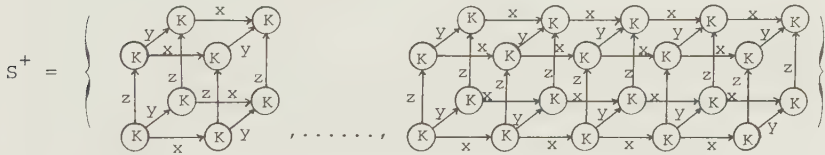
Die Syntaxanalyse der Schaltelementgraphen aus Beispiel 24 benötigt pro Graph Werte um durchschnittlich vier Sekunden, wobei auffällt, daß sie für Graphen, in deren Ableitungsfolge mehrere hohe Produktionsnummern auftreten, deutlich länger dauert als für Graphen, die durch Produktionen mit niedrigen Nummern erzeugbar sind.

Hieraus ist ablesbar, daß der Umfang der zugrundeliegenden Grammatik bei Aufgabenstellungen wie der Analyse von Schaltelementgraphen, bei denen die Abarbeitung von Sackgassen wegen der beschränkten Ansatzmöglichkeiten für Produktionen nur eine geringe Rolle spielt, einen wesentlichen Einfluß auf die durchschnittlichen Analysezeiten ausüben kann.

Des weiteren ist feststellbar, daß jeder Widerstandsgraph durch drei und jeder Diodengraph durch vier Produktionen ableitbar ist und die Grammatiken auch nicht in der Stichprobe enthaltene Graphen akzeptieren, wenn sich diese aus Teilstücken von in der Stichprobe enthaltenen Graphen zusammensetzen lassen. Hieraus kann man den Schluß ziehen, daß es bei der Erstellung einer Stichprobe im allgemeinen ausreicht, jedes mögliche Teilstück zwischen zwei Ansatzpunkten einmal in der Stichprobe enthalten zu haben und daß alle Neukombinationen von bekannten Teilstücken unabhängig von ihrem Vorhandensein in der Stichprobe zur Sprache der inferierten Grammatik gehören.

Beispiel 29:

Zur Feststellung, wie sehr die Analysezeiten von der Länge der Ableitungsfolge abhängig sein können, wurde eine Stichprobe von ein bis vier aneinandergereihten Würfeln der folgenden Form:



dem Inferenzverfahren mit Gedächtnis eingegeben. Die dabei resultierende Grammatik enthält insgesamt dreizehn Produktionen und bleibt vom dritten Stichprobenelement an unverändert.

Die Analysezeiten betragen für das

1. Stichprobenelement mit 7 Produktionen 7 Sek.
2. Stichprobenelement mit 12 Produktionen 42 Sek.
3. Stichprobenelement mit 17 Produktionen 265 Sek.
4. Stichprobenelement mit 22 Produktionen 1215 Sek.

Hieraus ist ablesbar, daß die Analysezeiten mit der Länge der Ableitungsfolgen ziemlich schnell anwachsen und es deshalb auch vom Analyseaufwand her gesehen wesentlich günstiger ist, einen Graphen aus wenigen großen statt aus vielen kleinen Produktionen zusammenzusetzen.

5. ERWEITERUNG DES ANSATZES AUF PROGRAMMIERTE GRAPHGRAMMATIKEN

In diesem Kapitel wird die Erweiterung des in Kapitel 3 und 4 geschilderten Ansatzes auf die Inferenz von programmierten Graphgrammatiken vorgestellt und ein Syntaxanalyseverfahren für programmierte Graphgrammatiken beschrieben. Die durch diese Maßnahmen erreichbaren Verbesserungen werden durch Beispiele illustriert.

5.1 INFERENZ VON PROGRAMMIERTEN GRAPHGRAMMATIKEN

Die teilweise auf den ersten Blick erkennbaren Gesetzmäßigkeiten in den Produktionsfolgen zur Erzeugung von Stichprobenelementen in Beispiel 27 oder 28 legen es nahe, die Vorteile einer Erweiterung des bisherigen Ansatzes auf programmierte Graphgrammatiken [Bu 79] zu untersuchen.

Die Grundidee der Programmierung einer Grammatik [Ro 69] besteht darin, die Reihenfolge, in der die Produktionen angewendet werden dürfen, dadurch zu reglementieren, daß man zu jeder Produktion angibt, welche Produktionen nach ihrer erfolgreichen bzw. fehlgeschlagenen Anwendung ausgeführt werden dürfen. Die Notation dieses zusätzlichen Formalismus kann entweder in Form einer Tabelle [Ro 69] oder in Form eines Kontrolldiagramms [Bu 79] erfolgen; der größeren Anschaulichkeit wegen soll hier hauptsächlich von der zweiten Möglichkeit Gebrauch gemacht werden.

Die in der Programmierung niedergelegte Information über die Aufeinanderfolge von Produktionen läßt sich im vorliegenden Fall aus der Analyse der Produktionsfolgen zur Erzeugung der Stichprobenelemente gewinnen, wie sie entweder gleich beim Inferenzverfahren registriert oder bei einer Syntaxanalyse aller Stichprobenelemente ermittelt werden können. Diese Produktionsfolgen erlauben jedoch nur Aussagen über das weitere Vorgehen nach der erfolgreichen Anwendung einer bestimmten Produktion und enthalten keine Information darüber, was im

Falle der Nichtanwendbarkeit einer Produktion zu tun ist. Deshalb ist es naheliegend, die Aufgabenstellung der Konstruktion einer Programmierung für eine Graphgrammatik auf Aussagen über Folgeproduktionen nach der erfolgreichen Anwendung einer Produktion zu beschränken.

Für die Formalismen zur Notation der Aufeinanderfolge von Produktionen bedeutet dies, daß eine Tabelle keine Mengen von Folgeproduktionen bzw. ein Kontrolldiagramm keine Kanten für den Mißerfolgsfall mehr enthält, wodurch sich die zur Beschreibung der Programmierung einer Graphgrammatik erforderlichen Definitionen folgendermaßen vereinfachen lassen:

Definition 15:

Ein Kontrolldiagramm K zur Darstellung der Programmierung einer Graphgrammatik $G = (V_N, V_E, V_{NT}, V_{ET}, X_O, P)$ ist ein (nicht notwendigerweise schlingenfrier) gerichteter Graph über $(V_P \cup \{A, Z\}, \emptyset)$ mit folgenden Eigenschaften:

- a) Der Graph K enthält genau einen mit A markierten Anfangsknoten und genau einen mit Z markierten Endknoten. Diese beiden Knoten sind schlingenfrier. Der Anfangsknoten besitzt nur auslaufende Kanten und der Endknoten nur einlaufende Kanten.
- b) Alle übrigen Graphknoten sind verschieden markiert, wodurch die Knotenmarkierungsfunktion m_N injektiv wird, und jeder Knotenmarkierung $l_i \in V_P$ kann über eine injektive Abbildung $f: V_P \rightarrow P$ eine der Produktionen $p_i \in P$ zugeordnet werden.

Der Einfachheit halber sollen die Markierungen der Knoten des Kontrolldiagramms außer bei A und Z im folgenden aus der Nummer der entsprechenden Produktion in P bestehen (d.h. $f^{-1}(p_i) = i$).

Definition 16:

- a) Eine Graph-Satzform X_{i+1} ist aus einer durch die Anwendung der Produktionsfolge $p_1 \dots p_i$ entstandenen Graph-Satzform X_i mittels der Produktion p_j programmiert ableitbar (abgek.

$X_i \xrightarrow[\text{pr}]{p_j} X_{i+1}$) gdw. in G $X_i \xrightarrow{p_j} X_{i+1}$ und in K eine Kante

$(m_N^{-1}(i), m_N^{-1}(j))$, d.h. eine Kante von einem mit i markierten Knoten zu einem mit j markierten Knoten vorhanden ist.

- b) Die von einer programmierten Graphgrammatik erzeugbare Sprache ist

$$L(G_{pr}) = \{X | X \in L(G) \wedge \exists \text{ in } K \text{ ein gerichteter Kantenzug} \\ (m_N^{-1}(A), m_N^{-1}(1), m_N^{-1}(2), \dots, m_N^{-1}(n), m_N^{-1}(Z)), \\ \text{wobei } p_1 p_2 \dots p_n \text{ eine Produktionsfolge zur Er-} \\ \text{zeugung von } X \text{ durch } G \text{ ist}\}$$

Der Algorithmus zur Konstruktion eines Kontrolldiagramms gemäß Definition 15 läßt sich nun wie folgt beschreiben:

Gegeben sei eine Menge von Produktionen $P = \{p_1, p_2, \dots, p_n\}$ einer Graphgrammatik G , die vom Inferenzverfahren nach der Verarbeitung einer Stichprobe $S^+ = \{S_1, S_2, \dots, S_k\}$ geliefert wurde. Bekannt seien außerdem die Produktionsfolgen $p_{i1} p_{i2} \dots p_{il}$ zur Erzeugung aller Stichprobenelemente $S_i \in S^+$.

1. Initialisierung:

Das Kontrolldiagramm K besteht anfangs aus $|P| + 2$ isolierten Knoten mit den Markierungen $A, Z, 1, 2, \dots, n$.

2. Verarbeitungszyklus:

Einlesen der nächsten Produktionsfolge $p_{i1} p_{i2} \dots p_{il}$. Falls in K die entsprechenden Kanten nicht schon vorhanden sind, Ergänzung von K um die Kanten

$$\begin{aligned} &- (m_N^{-1}(A), m_N^{-1}(i_1)) \\ &- (m_N^{-1}(i_1), m_N^{-1}(i_2)) \\ &\quad \vdots \\ &- (m_N^{-1}(i_l), m_N^{-1}(Z)) . \end{aligned}$$

Falls noch weitere Produktionsfolgen zu verarbeiten sind, Wiederholung von 2., ansonsten Sprung nach 3.

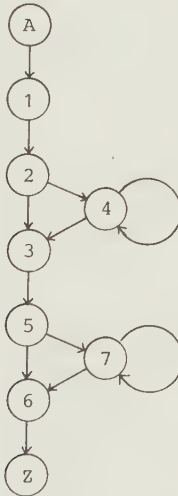
3. Abschluß:

Entfernung aller isolierten Knoten aus dem Graphen K und Ausgabe des Kontrolldiagramms.

Beispiel 30:

Für die Grammatik G_1 aus Beispiel 22 liefert die Syntaxanalyse in Beispiel 27 die Produktionsfolgen 1 2 3 5 6, 1 2 4 3 5 7 6 und 1 2 4 4 3 5 7 7 6 zur Erzeugung der drei Stichprobenelemente aus Beispiel 22. Der beschriebene Algorithmus zur Konstruktion eines Kontrolldiagramms erzeugt zuerst neun isolierte Knoten mit den Markierungen A, 1, 2, 3, 4, 5, 6, 7, Z und dann für die drei Produktionsfolgen die im untenstehenden Graphen eingezeichneten Verbindungskanten:

K_1 :



Nachdem alle in G vorhandenen Produktionen in mindestens einer Produktionsfolge zur Erzeugung eines Stichprobenelements vorkommen, enthält das Kontrolldiagramm keine isolierten Knoten und der Konstruktionsprozeß ist beendet.

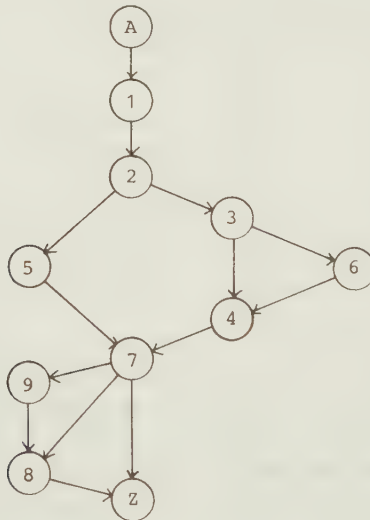
Wie man aus dem Algorithmus und aus dem vorangegangenen Beispiel ersieht, ist die Programmierung einer Grammatik gemäß Definition 15 sehr leicht und schnell zu ermitteln. Die Registrierung der möglichen Paare von Folgeproduktionen bereits während des Inferenzprozesses verursacht gemessen an den Rechenzeiten für den Inferenzalgorithmus praktisch keinen nen-

nenswerten Mehraufwand, liefert aber nicht immer das bestmögliche Ergebnis (siehe Beispiel 31). Die Ermittlung der Produktionsfolgen zur Erzeugung der Stichprobe während eines Syntaxanalyselaufs benötigt zwar noch einmal Rechenzeit, die in der gleichen Größenordnung wie die des Inferenzalgorithmus liegt, bietet aber dafür die Chance, redundante Produktionen oder nicht mit der minimalen Anzahl von Produktionen zum Ziel führende Ableitungen zu eliminieren.

Beispiel 31:

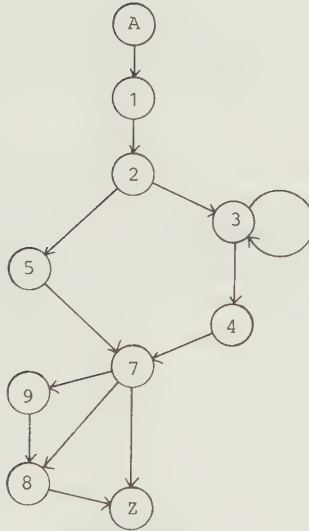
Gegeben sei die Grammatik G_{1010} aus Beispiel 25. Die während des Inferenzprogramms registrierten Produktionsfolgen zur Erzeugung der drei Häuschen aus Beispiel 22 sind 1 2 5 7, 1 2 3 4 7 8 und 1 2 3 6 4 7 9 8, woraus sich folgendes Kontrolldiagramm K_2 konstruieren läßt:

K_2 :



Die von einer Syntaxanalyse der Stichprobe ermittelten Produktionsfolgen für die gleiche Grammatik und die gleiche Stichprobe sind 1 2 5 7, 1 2 3 4 7 8 und 1 2 3 3 4 7 9 8. Hieraus ergibt sich folgendes Kontrolldiagramm K_3 :

K_3 :



Bei der Konstruktion von K_3 wurde die Produktion 6 als redundant erkannt und eliminiert.

Im Zusammenhang mit Beispiel 30 und 31 ist erwähnenswert, daß nicht jeder gerichtete Kantenzug vom Anfangs- zum Endknoten eines Kontrolldiagramms der Generierung eines terminalen Graphen entsprechen muß, wovon man sich anhand der Produktionsfolge 1 2 4 4 3 5 7 6 in Beispiel 30 leicht überzeugen kann.

Aus Beispiel 30 ist auch ersichtlich, daß dem Instrument der Programmierung die Fähigkeit fehlt, Schleifendurchläufe im Kontrolldiagramm zählen und sie mit späteren Schleifendurchläufen in Relation setzen zu können, wie dies z.B. bei Grammatiken mit Kontrollmenge [GiSp 68] der Fall ist. Der Grund für dieses Unvermögen besteht in der Tatsache, daß man die Konstruktion eines Kontrolldiagramms nach Definition 15 mit der Konstruktion eines endlichen Automaten oder einer regulären Grammatik für eine Menge von Zeichenketten über dem Alphabet P vergleichen kann und von endlichen Automaten bzw. regulären Grammatiken bekannt ist, daß sie sich zur

Modellierung von Sprachen der Form " uv^nwx^ny ", wie sie zur Darstellung der Produktionsfolgen aus Beispiel 30 ideal wären, nicht eignen.

Dieser letzte Vergleich zeigt auch, daß es sich bei dem geschilderten Problem der Konstruktion eines Kontrolldiagramms im Grunde um ein Inferenzproblem allereinfachster Art handelt, und daß man bei eventuellen weiteren Untersuchungen in Richtung mächtigere Kontrollmechanismen für die Steuerung der Aufeinanderfolge von Produktionen wahrscheinlich Anleihen bei konstruktiven Inferenzverfahren für kontextfreie Zeichenkettengrammatiken oder "augmented transition networks" machen kann.

5.2 SYNTAXANALYSE MIT PROGRAMMIERTEN GRAPHGRAMMATIKEN

Das hauptsächliche Ziel bei der Einführung programmierter Graphgrammatiken ist die Einschränkung der Vielzahl von Möglichkeiten, mit einer nichtprogrammierten Graphgrammatik Satzformen generieren zu können, die entweder terminal und den Stichprobenelementen wenig ähnlich oder nicht zu einem terminalen Graphen ableitbar und für die Syntaxanalyse nur hinderlich sind. Dies bedeutet, daß die Programmierung zu einer Verkleinerung der Anzahl der über die Stichprobe hinaus generierbaren Graphen und zu einer Reduzierung der Syntaxanalysezeiten insbesondere für nicht erzeugbare Graphen führen soll.

Um das Ausmaß dieser beiden prognostizierten Wirkungen der Programmierung beurteilen zu können, wurde ein zweiter Syntaxanalysealgorithmus entwickelt, der das Kontrolldiagramm einer Grammatik in die Suche nach der nächsten anwendbaren Produktion miteinbezieht.

Dieser neue Syntaxanalysealgorithmus hat im wesentlichen den gleichen Aufbau wie der im Struktogramm von Bild 8 dargestellte; die hauptsächlichen Unterschiede liegen in der Vorgehensweise bei der Suche nach der nächsten anwendbaren Produktion. Beim Syntaxanalyseverfahren von Kapitel 4 bildeten die in der Graphsatzform vorhandenen nichtterminalen Knoten die Ausgangsbasis

für den Vergleich mit den linken Seiten aller Ketten- bzw. Sternproduktionen. Beim Syntaxanalyseverfahren für programmierte Graphgrammatiken wird als erstes im Kontrolldiagramm nachgesehen, welche Folgeproduktionen möglich sind, und dann wird nur noch deren Anwendbarkeit auf die vorhandene Graph-Satzform untersucht.

Die wesentlichen Einzelheiten des Verfahrens sind in dem folgenden Struktogramm dargestellt, wobei die Unterschiede zum Syntaxanalyseverfahren von Kapitel 4 durch stärkere Umrandungen hervorgehoben sind:

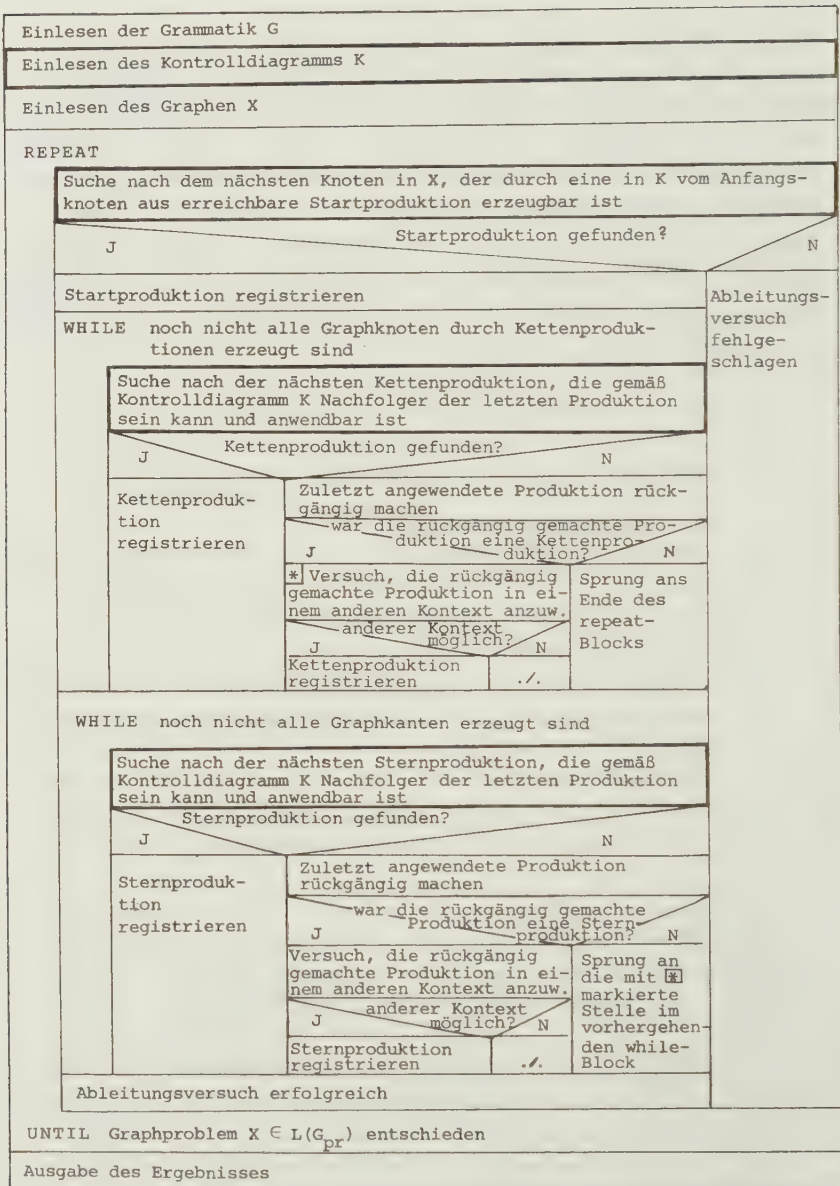


Bild 9 Struktogramm der Syntaxanalyse eines Graphen X mit Hilfe
einer programmierten Graphgrammatik

Das Problem der Reihenfolge, in der die möglichen Folgeproduktionen auf ihre Anwendbarkeit hin überprüft werden, spielt hier eine weit weniger wichtige Rolle als in Kapitel 4, weil die Anzahl der Alternativen beim Ableitungsprozeß durch die Programmierung so stark eingeschränkt wird, daß sich Veränderungen der Reihenfolge der Überprüfung von Folgeproduktionen nur an sehr wenigen Stellen bei der Analyse bemerkbar machen. Versuche, die Reihenfolge nach denselben Ordnungskriterien wie in Kapitel 4.2 zu variieren, brachten keine signifikanten Unterschiede bezüglich der Rechenzeiten mit sich und so wurde schließlich einer Sortierung der zu überprüfenden Alternativen nach abnehmender Anwendungshäufigkeit bei der Erzeugung der Stichprobe der Vorzug gegeben, weil dies die insgesamt stabilste Strategie zu sein scheint (siehe dazu auch Kapitel 6.2).

Die Frage, ob es sich lohnt, die Produktionsfolgen zur Erzeugung der Stichprobenelemente in einem gesonderten Syntaxanalyselauf zu ermitteln, ist abhängig davon, ob und wie sehr die Graphgrammatik und das Kontrolldiagramm sich dadurch vereinfachen lassen. Generell ist zu beobachten, daß die Chance, auf diese Weise redundante Produktionen eliminieren zu können, mit dem Umfang der Grammatik zunimmt und bei der Inferenzstrategie ohne Gedächtnis größer ist als bei der Inferenzstrategie mit Gedächtnis.

Das Ausmaß der zu Beginn dieses Kapitels prognostizierten Reduzierung des Sprachumfangs durch die Berücksichtigung der Programmierung läßt sich an dieser Stelle noch nicht quantitativ (siehe Kapitel 7), sondern nur anhand einiger beispielhafter Beobachtungen dokumentieren. Aus Definition 16b) und dem Verfahren zur Konstruktion des Kontrolldiagramms folgt unmittelbar, daß für die von der programmierten Graphgrammatik erzeugte Sprache $L(G_{pr})$ die beiden Beziehungen

$$L(G_{pr}) \subseteq L(G) \quad \text{und} \quad S^+ \subseteq L(G_{pr})$$

gelten, aber es ist ohne Durchführung weitergehender Untersuchungen schwer abschätzbar, in welchem Zahlenverhältnis die drei Mengen S^+ , $L(G_{pr})$ und $L(G)$ zueinander stehen.

Beispiel 32:

Die Grammatik G_{1010} aus Beispiel 25 ist imstande, Verallgemeinerungen der Stichprobe aus Beispiel 22 in Form von Häuschen mit vier oder mehr Stockwerken zu generieren. Dieselbe Grammatik mit einer aus dieser Stichprobe konstruierten Programmierung (z.B. K_3 in Beispiel 31) ist dazu nicht mehr in der Lage, weil das Kontrollldiagramm die beiden für den rekursiven Anbau eines weiteren Stockwerks notwendigen Schlingen an der Kettenproduktion 3 und der Sternproduktion 9 aus den drei Ableitungsfolgen zur Erzeugung der Stichprobenelemente noch nicht "gelernt" hat. Abhilfe könnte geschaffen werden durch die Aufnahme eines vierstöckigen Häuschens in die Stichprobe, wodurch sich G_{1010} nicht verändern, aber das Kontrollldiagramm K_3 um die fehlende Schlinge erweitert würde.

Beispiel 32 zeigt, daß unter den durch die programmierte Grammatik G_{pr} nicht mehr erzeugbaren Graphen $X \in L(G) \setminus L(G_{pr})$ auch wünschenswerte Verallgemeinerungen der Stichprobe sein können, wenn man bei der Auswahl der Stichprobenelemente zu berücksichtigen vergißt, daß sich Rekursionsmöglichkeiten in der Programmierung i.a. einen Schritt später manifestieren als in der Grammatik. Deswegen sollte man bei der Inferenz einer programmierten Graphgrammatik die Stichprobe lieber etwas umfangreicher wählen und das Vorhandensein erwünschter Rekursionen im Kontrollldiagramm nachprüfen.

Als letzte und wichtigste Frage bleibt noch zu klären, welche Rechenzeitvorteile die Verwendung einer programmierten Graphgrammatik im Vergleich zu einer nichtprogrammierten bringt. Die erste Überraschung dabei ist, daß die Programmierung die Analyse nicht in jedem Fall beschleunigen muß, sondern ab und zu sogar geringfügig verzögern kann. Erklärungen dafür sind zum einen, daß sich der neben dem Suchaufwand zu investierende Verwaltungsaufwand durch die zusätzliche Programmierung erhöht, und zum anderen, daß die Sortierung der Produktionen nach fallender Anwendungshäufigkeit in Einzelfällen trotz der Programmierung schlechter sein kann als die Sortierung nach abnehmender Größe der rechten Seiten in Kapitel 4. Im allge-

meinen ist jedoch schon ein deutlicher Einsparungseffekt zu beobachten, der abhängig von der Verzweigungsstruktur des Kontrollldiagramms für Stichproben, die hauptsächlich aus durch die programmierte Grammatik erzeugbaren Graphen bestehen, typischerweise zwischen zwanzig und vierzig Prozent liegt.

Beispiel 33:

Eine Stichprobe bestehend aus fünf ein- bis fünfstöckigen Häuschen wie in Beispiel 22 ist unter Verwendung der nichtprogrammierten Grammatik G_{1010} in 27 Sekunden und unter Verwendung einer nach dem Vorschlag in Beispiel 32 erweiterten programmierten Grammatik in 13 Sekunden analysierbar.

Die eindruckvollsten Rechenzeiteinsparungen durch Verwendung einer programmierten Grammatik erzielt man bei der Analyse von durch die Grammatik nicht erzeugbaren Graphen, deren Analysezeiten die beste Vorstellung von der Anzahl der insgesamt auszubprobierenden Alternativen vermitteln. Bei Graphen mit circa fünfzehn Knoten und zwanzig Kanten, die sich nur in wenigen, aber ausschlaggebenden Details von Stichprobenelementen unterscheiden, dauert die Feststellung dieses Sachverhalts mit Hilfe einer nichtprogrammierten Grammatik oft ein bis zwei Minuten, wohingegen die programmierte Grammatik meist in weniger als zehn Sekunden zum gleichen Ergebnis führt.

Zusammenfassend läßt sich feststellen, daß die Inferenz einer programmierten Graphgrammatik kaum mehr Aufwand erfordert als die Inferenz einer nichtprogrammierten. Durch die Ausnutzung der in der Programmierung enthaltenen Information wird i.a. der Umfang der durch die inferierte Grammatik erzeugbaren Sprache eingeschränkt und die Analysezeiten können bis auf die Hälfte reduziert werden. Infolgedessen kann die Programmierung als äußerst einfaches und wirkungsvolles Instrument zur Verbesserung des Inferenzergebnisses und des Syntaxanalyseverfahrens angesehen werden.

6. ERWEITERUNGSMÖGLICHKEITEN DES ANSATZES AUF STOCHASTISCHE GRAPHGRAMMATIKEN UND STOCHASTISCHE PROGRAMMIERUNG

Das folgende Kapitel beschreibt die Möglichkeiten und Schwierigkeiten, neben der in den Produktionen und der Programmierung enthaltenen Strukturinformation auch statistische Information über die Anwendungswahrscheinlichkeiten von Produktionen, Ableitungswahrscheinlichkeiten von Graphen oder Wahrscheinlichkeiten für die Verfolgung verschiedener Alternativen im Kontrolldiagramm zu gewinnen. Das Kapitel ist unterteilt in Abschnitt 6.1 zum Thema Inferenz von stochastischen Graphgrammatiken und Abschnitt 6.2 über stochastische Syntaxanalyse mit Hilfe von programmierten Graphgrammatiken.

6.1 INFERENZ VON STOCHASTISCHEN GRAPHGRAMMATIKEN

Eine Stichprobe von Mustern enthält im allgemeinen nicht nur Strukturinformation über den Aufbau der Muster, sondern auch statistische Information über die Vorkommenshäufigkeit bestimmter Musterstrukturen. Im vorliegenden Fall ist die Strukturinformation bereits in den Produktionen und der Programmierung der inferierten Grammatiken enthalten. Die bisher noch weitgehend unberücksichtigt gebliebene statistische Information ist in das Strukturmodell Grammatik miteinbeziehbar durch den Übergang zu stochastischen Grammatiken.

Definition 17:

Eine *stochastische Graphgrammatik* ist ein Tupel $G_s = (V_N, V_E, V_{NT}, V_{ET}, X_O, P_s)$ mit V_N, V_E, V_{NT}, V_{ET} und X_O wie in Definition 9. P_s ist eine Menge von stochastischen Graphproduktionen der Form

$$q_i : X_1 \Rightarrow X_r$$

mit X_1 und X_r gemäß Definition 9 und q_i als Wahrscheinlichkeit einer Anwendung der Produktion $X_1 \Rightarrow X_r$. Für die Wahrscheinlichkeiten q_i muß gelten:

$$0 \leq q_i \leq 1 \quad \text{und} \quad \sum q(X_1 \Rightarrow \dots) = 1 \quad ,$$

d.h. die Summe der Wahrscheinlichkeiten aller Produktionen mit X_1 als linker Seite muß 1 ergeben.

Die von einer stochastischen Graphgrammatik erzeugte Sprache $L(G_s)$ umfaßt die gleiche Menge von terminalen Graphen wie die Sprache der gleichen Grammatik ohne Produktionswahrscheinlichkeiten und bietet zusätzlich die Möglichkeit, jedem terminalen Graphen seine Ableitungswahrscheinlichkeit in G_s zuzuordnen.

Die Berechnung der Ableitungswahrscheinlichkeit $q(X)$ eines terminalen Graphen X mit einer stochastischen Graphgrammatik G_s läßt sich analog zum Zeichenkettenfall [Fu 72] folgendermaßen durchführen:

- a) Die Ableitung eines Graphen $X \in L(G_s)$ durch die Produktionsfolge $p_1 p_2 \dots p_n$ hat die Wahrscheinlichkeit

$$q'(X) = q(p_1) \cdot q(p_2 | p_1) \cdot \dots \cdot q(p_n | p_1 p_2 \dots p_{n-1}) .$$

- b) Falls es in G_s insgesamt m verschiedene Möglichkeiten zur Erzeugung des Graphen X gibt, dann hat die Ableitung von X in G_s die Wahrscheinlichkeit

$$q(X) = \sum_{i=1}^m q'(X) .$$

- c) Unter der Voraussetzung, daß die Anwendung einer Produktion nur vom Vorhandensein der entsprechenden linken Seite und nicht von deren speziellem Erzeugungsprozeß abhängt, läßt sich die Berechnung von $q'(X)$ vereinfachen zu:

$$q'(X) = q(p_1) \cdot q(p_2) \cdot \dots \cdot q(p_n) .$$

- d) Bei eindeutigen Graphgrammatiken gibt es immer nur genau eine Möglichkeit, einen bestimmten Graphen $X \in L(G_s)$ abzuleiten. Falls also G_s eindeutig ist, kann man die Berechnung von $q(X)$ ohne Summation über die verschiedenen Ableitungsmöglichkeiten nach der folgenden Formel durchführen:

$$q(X) = q(p_1) \cdot q(p_2) \cdot \dots \cdot q(p_n) .$$

Aus der Spezifikation des Ableitbarkeitsbegriffs für die inferierten Graphgrammatiken in Definition 13 ist ersichtlich, daß die unter Punkt c) gemachte Voraussetzung für die betrachteten Graphgrammatiken erfüllt ist und die angegebene

Vereinfachung der Berechnung von $q'(X)$ für das weitere Vorgehen benutzt werden darf.

Problematischer ist es hingegen um die Eindeutigkeit der inferierten Graphgrammatiken bestellt. Die Erfahrung mit den Inferenzverfahren zeigt, daß sowohl eindeutige als auch nicht eindeutige Grammatiken erzeugt werden und daß insbesondere die größeren Grammatiken dazu tendieren, nicht eindeutig zu sein. Deshalb wird man von der unter Punkt d) geschilderten Vereinfachung der Berechnung von $q(X)$ nur in Ausnahmefällen Gebrauch machen können.

Eine sehr wünschenswerte Eigenschaft von stochastischen Grammatiken ist ihre Konsistenz:

Definition 18:

Die Zuordnung der Produktionswahrscheinlichkeiten q_i zu den Produktionen einer Grammatik heißt *konsistent*, wenn die Beziehung

$$\sum_{X \in L(G_S)} q(X) = 1$$

erfüllt ist.

Verfahren zur Überprüfung der Konsistenz von unendlichen Sprachen gibt es leider nur für stochastische reguläre und kontextfreie Zeichenkettensprachen [BoTh 73]; über die Entscheidbarkeit dieses Problems für kontextsensitive Zeichenkettensprachen oder Graphsprachen ist bisher nichts bekannt. Dies bedeutet, daß man die Konsistenz einer inferierten stochastischen Graphgrammatik nur in speziellen, besonders einfach gelagerten Fällen effektiv nachprüfen kann. Zu diesen Spezialfällen zählen stochastische Graphgrammatiken, die einer kontextfreien Zeichenkettensprache entsprechen, Graphgrammatiken, die eine endliche Sprache erzeugen und Graphgrammatiken mit so wenigen und leicht überschaubaren Rekursionsmöglichkeiten, daß man die Summe der $q(X)$ über einen Ausdruck mit unendlichen geometrischen Reihenentwicklungen berechnen kann.

Trotz der zu erwartenden Schwierigkeiten mit Grammatiken, deren

Eindeutigkeit und Konsistenz nicht ohne weiteres vorausgesetzt werden können, soll im folgenden geschildert werden, wie man Produktionswahrscheinlichkeiten aus der Stichprobe gewinnen kann.

Zur Inferenz von nicht stochastischen Grammatiken ist es ausreichend, wenn jeder durch die Grammatik zu erzeugende Satz der Sprache in der Stichprobe einmal vorkommt, weil weitere Vorkommen desselben Satzes die Zwischengrammatik unverändert lassen und nur zusätzlichen Aufwand verursachen. Nachdem es bei der Inferenz von stochastischen Grammatiken aber nun auf die Häufigkeiten des Auftretens von Musterstrukturen ganz entscheidend ankommt, müssen mehrfache Vorkommen desselben Satzes der Sprache berücksichtigt und in die Wahrscheinlichkeitsberechnung miteinbezogen werden. Die kompakteste und am effizientesten zu verarbeitende Darstellung für diesen Sachverhalt ist die sogenannte *Experimentmenge*

$$E = \{ (X_i, c_i) \mid X_i \in S^+ \wedge c_i \in \mathbb{N} \},$$

durch die jedem Stichprobenelement $X_i \in S^+$ seine Vorkommenshäufigkeit in der Stichprobe c_i zugeordnet wird.

Beispiel 34:

Gegeben sei die Experimentmenge

$$E = \{ (\textcircled{K} \xrightarrow{A} \textcircled{K} \xrightarrow{B} \textcircled{K} , 4) \\ (\textcircled{K} \xrightarrow{A} \textcircled{K} \xrightarrow{B} \textcircled{K} \xrightarrow{A} \textcircled{K} \xrightarrow{B} \textcircled{K} , 2) \\ (\textcircled{K} \xrightarrow{A} \textcircled{K} \xrightarrow{B} \textcircled{K} \xrightarrow{A} \textcircled{K} \xrightarrow{B} \textcircled{K} \xrightarrow{A} \textcircled{K} \xrightarrow{B} \textcircled{K} , 1) \} ,$$

die sich von der Stichprobe aus Beispiel 20 nur durch die zusätzlichen Häufigkeiten c_i unterscheidet. Will man nun die Grammatik aus Beispiel 21 zu einer stochastischen Grammatik erweitern, so ist dafür lediglich die Wahrscheinlichkeit q_3 für Produktion 3 zu bestimmen:

1. $q_1 = 1$: START $\Rightarrow$ $\boxed{K}$
2. $q_2 = 1 - q_3$: $\boxed{K} \Rightarrow (K) \xrightarrow{A} (K) \xrightarrow{B} (K)$
3. q_3 : $\boxed{K} \Rightarrow (K) \xrightarrow{A} (K) \xrightarrow{B} \boxed{K}$

Aus dieser höchst einfach strukturierten Grammatik ist außerdem ersichtlich, daß sie eindeutig ist und für die Ableitungswahrscheinlichkeit eines Graphen mit $2i + 1$ Knoten und $2i$ Kanten

$$q(X_{2i}) = q_1 \cdot q_3^{i-1} \cdot q_2 = q_3^{i-1} (1 - q_3)$$

gilt. In diesem Fall ist auch die Konsistenzbedingung

$$\sum_{i=1}^{\infty} q(X_{2i}) = \sum_{i=1}^{\infty} q_3^{i-1} (1 - q_3) = 1$$

leicht überprüfbar.

Zur Lösung der Aufgabe, für eine Grammatik mit k Produktionen, die l verschiedene linke Seiten besitzen, die $k - 1$ unbekannten Produktionswahrscheinlichkeiten zu bestimmen, kann man folgenden Ansatz [FuBo 75] machen:

- Unter der Voraussetzung, daß das Auftreten eines Stichprobenelements unabhängig von den übrigen Stichprobenelementen erfolgt, ist die Wahrscheinlichkeit für die Beobachtung der Experimentmenge $E = \{(X_1, c_1), (X_2, c_2), \dots, (X_n, c_n)\}$

$$q(E) = \prod_{i=1}^n q(X_i)^{c_i}.$$

- Für den Fall, daß die Grammatik eindeutig ist, lassen sich die $q(X_i)$ berechnen aus

$$q(X_i) = \prod_{j=1}^k q(p_j)^{a_j(X_i)}$$

mit $a_j(X_i)$ als Häufigkeit des Vorkommens von Produktion p_j in der Ableitung von X_i .

- Zur Bestimmung der Maximum-Likelihood-Schätzwerte [Kr 65] der unbekannten Parameter $q(p_j)$ müssen die Lösungen eines Gleichungssystems der Form

$$\frac{\partial \ln q(E)}{\partial q(p_j)} = 0 \quad \text{für } j = 1, 2, \dots, k$$

unter Berücksichtigung der Nebenbedingungen, daß die Summe der $q(p_j)$ mit gleicher linker Seite immer 1 ergeben muß, berechnet werden.

- Unter der Voraussetzung, daß die Grammatik eindeutig ist, kann dieses Gleichungssystem auf direktem Wege [MaBo 77] gelöst werden und man erhält als Maximum-Likelihood-Schätzwert für die Produktionswahrscheinlichkeiten:

$$\hat{q}(p_j) = \frac{a_j(E)}{a_{x_{1j} \Rightarrow \dots}(E)} \quad (*)$$

als Quotient aus der Häufigkeit der Anwendung von Produktion j zur Erzeugung von E und der Häufigkeit der Anwendung einer Produktion mit gleicher linker Seite wie Produktion j ebenfalls zur Erzeugung von E .

Dieses letzte Ergebnis ist intuitiv verständlich und ohne großen Aufwand entweder im Anschluß an das Inferenzverfahren oder auch im Anschluß an einen Syntaxanalyselauf mit der um die Häufigkeiten c_i ergänzten Stichprobe berechenbar.

Man braucht dafür lediglich zu registrieren, wie oft jede Produktion zur Erzeugung von E angewendet wird und diese Zahlen $a_j(E)$ durch die Summe aller $a_{1j}(E)$ mit gleicher linker Seite wie Produktion j zu dividieren.

Beispiel 35:

Die Registrierung der Anwendungshäufigkeiten der Produktionen aus Beispiel 34 zur Erzeugung der Experimentmenge aus Beispiel 34 ergibt:

$$a_1(E) = 7$$

$$a_2(E) = 7$$

$$a_3(E) = 4$$

Nachdem Produktion 2 und 3 die gleiche linke Seite besitzen, erhält man

$$q_1 = 1$$

$$q_2 = \frac{7}{11}$$

$$q_3 = \frac{4}{11}$$

als Maximum-Likelihood-Schätzwerte für die Produktionswahrscheinlichkeiten.

Bei kontextsensitiven Produktionen verdient der Aspekt der Zusammenfassung der $a_{1j}(E)$ mit gleicher linker Seite eine zusätzliche Bemerkung. Es ist zwar auch hier nicht schwierig, Produktionen mit der gleichen Menge von Kontextknoten auf der linken Seite herauszufinden und deren Anwendungshäufigkeiten zusammenzuzählen, aber man erfaßt damit im Gegensatz zu kontextfreien Produktionen im allgemeinen nicht alle möglichen Folgeproduktionen, weil es auch noch Produktionen mit $X_{1k} \subsetneq X_{1j}$, d.h. mit einer echten Teilmenge der Kontextknoten von X_{1j} auf der linken Seite geben kann, die genauso gut wie die Produktionen mit der linken Seite X_{1j} anwendbar sind. Eine Mitberücksichtigung dieser Produktionen würde das Problem aber auch nicht lösen, weil dann die Relation

$$r_{jk} \triangleq \text{Produktion } j \text{ anwendbar} \Rightarrow \text{Produktion } k \text{ anwendbar}$$

für Produktionen mit linken Seiten $X_{1k} \subsetneq X_{1j}$ unsymmetrisch ist und der für die Normalisierung der Wahrscheinlichkeitswerte erforderlichen Aufteilung der Produktionen in Äquivalenzklassen bezüglich r_{jk} im Wege steht.

Versuche, dieses Problem durch Produktionswahrscheinlichkeiten $q(p_j | p_1 p_2 \dots p_{j-1})$, deren Wert von der Vorgeschichte der vorhandenen Graph-Satzform abhängt, in den Griff zu bekommen, führen zu so umfangreichen Differentialgleichungssystemen für die Schätzwertbestimmung, daß dafür keine allgemeine Lösung angegeben werden kann.

Auf vergleichbare Schwierigkeiten stößt man bei dem Versuch, bei nicht eindeutigen Grammatiken die Wahrscheinlichkeit $p(E)$

aus Faktoren der Form

$$q(X_i) = \sum_{i=1}^m q'(X_i)$$

zusammensetzen. Hierbei kommt noch erschwerend hinzu, daß sich die dafür erforderliche Information über sämtliche Möglichkeiten, einen Graphen X_i mit einer Graphgrammatik abzuleiten, gerade bei nicht eindeutigen Grammatiken weder durch das Inferenzverfahren noch durch das Syntaxanalyseverfahren vollständig bestimmen läßt, weil beide Verfahren nur jeweils eine Ableitungsmöglichkeit für jeden Stichprobengraphen liefern.

Zusammenfassend muß man feststellen, daß die Inferenz von stochastischen Graphgrammatiken durch Berechnung des Maximum-Likelihood-Schätzwertes für die Produktionswahrscheinlichkeiten nur in besonders einfach gelagerten Fällen auf einer tragfähigen theoretischen Basis erfolgen kann und man nach der Gleichung (*) geschätzte Produktionswahrscheinlichkeiten für Grammatiken, deren Eindeutigkeit oder Konsistenz zweifelhaft sind, mit großer Vorsicht interpretieren und/oder weiterverwenden sollte.

6.2 SYNTAXANALYSE VON GRAPHGRAMMATIKEN MIT STOCHASTISCHER PROGRAMMIERUNG

Eine weitere Nutzenanwendung der in der Experimentmenge enthaltenen statistischen Information besteht darin, für die Programmierung einer Grammatik nicht nur die zur Erzeugung der Stichprobe notwendigen Paare von Folgeproduktionen, sondern gleichzeitig auch deren Häufigkeiten zu registrieren, um auf der Basis dieser zusätzlichen Information eine Verbesserung der Syntaxanalysestrategie vornehmen zu können.

Die diesem Ansatz zugrundeliegenden Ideen stammen von Huang und Fu [HuFu 72], die dieses Konzept für kontextfreie programmierte Zeichenkettengrammatiken untersucht und zur Klassifikation von Chromosomen angewendet haben. Das hauptsächliche Ergebnis dieser Arbeit ist, daß die mittlere Anzahl von Schritt-

ten zur Erkennung eines Satzes einer kontextfreien programmierten Grammatik, die als Maß zur Abschätzung des Aufwands für die Syntaxanalyse gelten kann, bei einer Syntaxanalysestrategie, die die möglichen Folgeproduktionen nach abnehmender Wahrscheinlichkeit geordnet überprüft, niedriger ist als bei anderen Syntaxanalysestrategien, die die nächste Folgeproduktion zufällig oder nach einer nicht nach Wahrscheinlichkeiten geordneten festen Reihenfolge auswählen.

Im vorliegenden Fall ist es deshalb interessant zu untersuchen, ob und wie sich die Wahrscheinlichkeitsverteilung der nach einer Produktion i anwendbaren Folgeproduktionen k aus der Experimentmenge schätzen läßt und ob sich die in [HuFu 72] gefundenen Resultate auch für Graphgrammatiken bestätigen.

Die Ermittlung der Folgeproduktionswahrscheinlichkeiten $q_{ij} = q(p_j | p_i)$ ist eine Aufgabe, die man als vollkommen analog zur Schätzung der Produktionswahrscheinlichkeiten $q_i = q(p_i)$ der stochastischen Grammatik in Kapitel 6.1 ansehen kann.

Der einzige Unterschied besteht darin, daß die Wahrscheinlichkeiten q_{ij} von der zuletzt ausgeführten Produktion i abhängig sind und es deshalb zu jeder Produktion j eine Vielzahl von Produktionen q_{ij} , q_{jj} , q_{kj} bzw. q_{j1} , q_{jm} usw. geben wird, die sich als Gewichtungen der ein- bzw. auslaufenden Kanten des Knotens j im Kontrolldiagramm gut darstellen lassen.

Für eindeutige programmierte Grammatiken ist es ähnlich wie in Kapitel 6.1 möglich, den Wert $q(E)$ aus Faktoren der Form

$$q(X) = q(p_1 | A) \cdot q(p_2 | p_1) \cdot \dots \cdot q(p_n | p_{n-1}) \cdot q(Z | p_n)$$

zusammenzusetzen und unter Berücksichtigung der Nebenbedingungen

$$\sum_j q(p_j | p_i) = 1 \quad \forall i \in V_K \cup \{A\} \wedge \forall j \in V_K \cup \{Z\}$$

zu maximieren. Der Maximum-Likelihood-Schätzwert für die Folgeproduktionswahrscheinlichkeiten q_{ij} ist

$$\hat{q}(p_j | p_i) = \frac{a_{ij}(E)}{\sum_{k \in V_K \cup \{Z\}} a_{ik}(E)} \quad (**)$$

mit $a_{ij}(E)$ als Häufigkeit des Auftretens der Produktionsfolge $i\ j$ zur Erzeugung der Experimentmenge E .

Beispiel 36:

Gegeben seien die Experimentmenge und die Grammatik aus Beispiel 34. Die Syntaxanalyse der Experimentmenge liefert:

4-mal die Produktionsfolge 1 2

2-mal die Produktionsfolge 1 3 2

1-mal die Produktionsfolge 1 3 3 2 .

Hieraus sind die Häufigkeiten des Auftretens von Folgeproduktionspaaren

$$a_{A1}(E) = 7$$

$$a_{12}(E) = 4$$

$$a_{13}(E) = 3$$

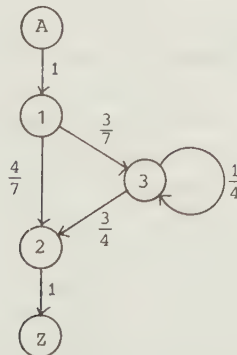
$$a_{2Z}(E) = 7$$

$$a_{32}(E) = 3$$

$$a_{33}(E) = 1$$

leicht registrierbar und die stochastische Programmierung der Grammatik läßt sich darstellen durch folgendes Kontrolldiagramm:

K_4 :



Für dieses Beispiel ist sogar die mittlere Anzahl von Schritten zur Erkennung eines Satzes der Sprache aus der stocha-

stischen Programmierung berechenbar:

$$\begin{aligned}
 AS &= \frac{4}{7} \cdot 3 + \sum_{n=0}^{\infty} \frac{3}{7} \cdot \left(\frac{1}{4}\right)^n \cdot \frac{3}{4} \cdot (4 + n) \\
 &= \frac{12}{7} + \frac{9}{7} \cdot \frac{1}{1 - \frac{1}{4}} + \frac{9}{28} \cdot \frac{\frac{1}{4}}{\left(1 - \frac{1}{4}\right)^2} \\
 &= 3,57 .
 \end{aligned}$$

Die Verwendung der Wahrscheinlichkeiten q_{ij} zur Berechnung der Ableitungswahrscheinlichkeit $q(X)$ eines Graphen X ist ähnlich problematisch wie die Verwendung der q_i aus Kapitel 6.1 zu diesem Zweck, weil

- a) durch identische Produktionsfolgen verschiedene terminale Graphen ableitbar sein können (siehe Beispiel 27)
- b) identische Graphen durch verschiedene Produktionsfolgen ableitbar sein können (siehe Beispiel 27) und
- c) die Überprüfung der Konsistenz der Sprache einer programmierten Graphgrammatik nur in besonders einfach gelagerten Fällen möglich ist (siehe [HuFu 72] und Kapitel 6.1).

Die Erfahrungen mit Veränderungen der Reihenfolge, in der die möglichen Folgeproduktionen überprüft werden, bestätigen das Ergebnis aus [HuFu 72] dahingehend, daß auch hier eine Sortierung der Alternativen nach abnehmender Wahrscheinlichkeit besser ist als irgendeine zufällige Reihenfolge. Der Vergleich dieser Strategie mit der in Kapitel 4 erfolgreich angewendeten Sortierung nach abnehmender Produktionsgröße brachte keine eindeutigen Vorteile für die eine oder andere Strategie und so kann man abschließend zumindest feststellen, daß die Berechnung der Folgeproduktionswahrscheinlichkeiten aus den Häufigkeiten ihres Vorkommens zur Erzeugung der Experimentmenge im Hinblick auf eine gute und stabile Syntaxanalysestrategie für programmierte Graphgrammatiken durchaus empfehlenswert ist.

7. GÜTEKRITERIEN ZUM VERGLEICH VON GRAPHGRAMMATIKEN

In diesem Kapitel wird untersucht, welche Möglichkeiten und Schwierigkeiten bei der Definition und Berechnung von Gütekriterien zum Vergleich von Graphgrammatiken zu erwarten sind. Das Kapitel ist unterteilt in Abschnitt 7.1 zum Thema strukturelle Gütekriterien und Abschnitt 7.2 zum Thema statistische Gütekriterien.

7.1 STRUKTURELLE GÜTEKRITERIEN

Die in dieser Arbeit vorgeschlagenen Inferenzverfahren erlauben aufgrund der Freiheitsgrade bei der Wahl der Parameter m und n bzw. bei der Wahl eines Inferenzverfahrens mit oder ohne Gedächtnis die Ableitung einer ganzen Reihe von Grammatiken mit oder ohne Programmierung für ein- und dieselbe Stichprobe. Diese Vielfalt von möglichen Problemlösungen erhöht einerseits die Chance des Anwenders, eine besonders gute Grammatik geliefert zu bekommen, belastet ihn aber andererseits mit der hauptsächlich bei umfangreicheren Grammatiken nicht leicht zu treffenden Auswahl einer für seine Zwecke geeignetsten Grammatik.

Um in dieser Situation Entscheidungshilfen bereitzustellen, versucht man, die für eine Anwendung wichtigen Eigenschaften einer Grammatik auf ein möglichst einfach berechenbares und vergleichbares Gütekriterium abzubilden. Die kompakteste Form für ein Gütekriterium ist die Berechnung einer Maßzahl für eine bestimmte Eigenschaft der Grammatik.

Die im vorliegenden Fall wesentlichsten Eigenschaften einer inferierten Graphgrammatik sind zum einen ihre Komplexität als Maß für den zu erwartenden Syntaxanalyseaufwand und zum anderen ihre Fähigkeit, die Stichprobe zu verallgemeinern, als Maß zur Kontrolle des angestrebten Lernerfolges.

Zur Bestimmung der Komplexität von Zeichenkettengrammatiken sind verschiedenste Maße in Gebrauch, darunter befinden sich

z.B. die Anzahl der Produktionen, die Größe des nichtterminalen Alphabets oder die maximale Länge einer Produktion [RoSn 72]; auch Kombinationen von mehreren dieser Maße oder die Summe der Länge aller Produktionen [Fe 72] sind für diesen Zweck verwendbar.

Zur Charakterisierung der Komplexität einer inferierten Graphgrammatik kommen von den genannten Komplexitätsmaßen nur die Anzahl der Produktionen und die Summe der Größe der rechten Seiten aller Produktionen in Frage. Nachdem bei der Syntaxanalyse wenige große Produktionen leichter zu überprüfen sind als eine nach dem zweiten Maß gleichwertige Menge von mehreren kleineren Produktionen, empfiehlt sich die Wahl von

$$K_p(G) = \text{Anzahl der Produktionen in } P$$

als Gütekriterium für den Vergleich der Komplexität zweier Graphgrammatiken.

Für den Vergleich des zu erwartenden Syntaxanalyseaufwands bei programmierten Graphgrammatiken spielt neben K_p auch die Verzweigungsstruktur der Kontrolldiagramme eine wichtige Rolle. Die Syntaxanalyse einer programmierten Graphgrammatik ist umso effizienter möglich, je weniger Alternativen das Kontrolldiagramm enthält, d.h. je weniger Kanten im Vergleich zur Anzahl der Knoten im Kontrolldiagramm vorhanden sind. Ein einfach zu überprüfendes Kriterium hierfür ist

$$K_K(G) = \frac{\text{Anzahl der Knoten im Kontrolldiagramm} - 1}{\text{Anzahl der Kanten im Kontrolldiagramm}} .$$

K_K hat einen Wert von 1, wenn zu jeder Produktion nur eine Folgeproduktion anwendbar ist, bzw. einen Wert umso kleiner als eins, je mehr Verzweigungsmöglichkeiten im Kontrolldiagramm enthalten sind.

Die Fähigkeit einer Grammatik, eine Stichprobe zu verallgemeinern, ist eine Eigenschaft, die man einer Grammatik im allgemeinen nicht "ansehen" oder durch einfache Rechenoperationen ermitteln kann. Die einzige Möglichkeit, einen quantitativen Vergleich der beiden Mengen S^+ und $L(G)$ durchzuführen, besteht darin, die Sprache $L(G)$ aufzuzählen bzw. falls $L(G)$ unendlich

ist, alle Graphen aus $L(G)$ bis zu einer bestimmten Größe aufzuzählen. Damit läßt sich dann ein Gütekriterium der Form

$$K_S(G) = \frac{\text{Anzahl verschiedener Graphen in } S^+}{\text{Anzahl verschiedener Graphen in } L(G) \leq \text{größtes Stichprobenelement } X_{\max}}$$

definieren, dessen Wert im Falle der Übereinstimmung der Mengen S^+ und $L(G, X_{\max}) = \{X | X \in L(G) \wedge X \leq X_{\max}\}$ den Wert 1 annimmt und im übrigen umso kleiner wird, je größer der Unterschied zwischen beiden Mengen ist. Zum Vergleich, ob ein terminaler Graph aus $L(G)$ kleiner oder gleich dem größten Stichprobenelement ist, sollte hier die Zahl der Knoten und die Zahl der Kanten herangezogen werden.

Die beiden Gütekriterien K_P und K_K sind ohne nennenswerten Aufwand im Anschluß an die Inferenz der Grammatik bzw. des Kontrolldiagramms berechenbar. Die Ermittlung des Wertes von K_S hingegen erfordert sowohl für nichtprogrammierte als auch für programmierte Graphgrammatiken ein Programm zur Aufzählung der Sprache einer beliebigen inferierten Graphgrammatik, das vom Umfang her mit der Syntaxanalyse vergleichbar ist (jeweils ca. 700 FORTRAN-Befehle) und von den Laufzeiten her die Syntaxanalyse-Programme um eine bis mehrere Größenordnungen übertrifft.

Deshalb wird es für größere Anwendungen unumgänglich sein, den Umfang der beiden zu vergleichenden Mengen S^+ und $L(G, X_{\max})$ dadurch zu reduzieren, daß man die Schranke für den größten zu vergleichenden Graphen niedriger ansetzt und das Kriterium zu

$$K_S'(G, X_s) = \frac{\text{Anzahl verschiedener Graphen in } S^+ \leq X_s}{\text{Anzahl verschiedener Graphen in } L(G) \leq X_s}$$

modifiziert. Die Erfahrungen mit diesem Kriterium K_S' zeigen, daß sich die für die Entscheidung zwischen zwei Grammatiken wichtige Eigenschaft $K_S(G_1) > K_S(G_2)$ bereits bei Graphen X_s , die die Anzahl der Graphen in der Stichprobe, die kleiner als X_s sind, auf relativ wenige Graphen reduzieren, im modifizierten Kriterium in Form von $K_S'(G_1, X_s) > K_S'(G_2, X_s)$ ab-

zeichnet und diese Tendenz bei Vergrößerung von X_S im allgemeinen erhalten bleibt. Dies bedeutet, daß es für den Vergleich der Verallgemeinerungsfähigkeit zweier Grammatiken in der Regel ausreicht, zwei Teilmengen $L(G_1, X_S)$ und $L(G_2, X_S)$ aufzuzählen und aus diesen die gewünschten Rückschlüsse zu ziehen.

Die Frage, wie groß X_S mindestens gewählt werden sollte, um einigermaßen repräsentative Ergebnisse zu erhalten, kann nur in bezug auf die gegebene Stichprobe und die dazugehörigen Grammatiken beantwortet werden. Die Zahl der Knoten und Kanten von X_S sollte jedenfalls die des kleinsten Stichprobenelements deutlich übersteigen und im Falle einer programmierten Grammatik wäre es günstig, wenn alle Bahnen vom A-Knoten zum Z-Knoten des Kontrolldiagramms bei der Erzeugung der Vergleichsteilmenge von Graphen mindestens einmal durchlaufen würden.

Der zur Berechnung des Gütekriteriums K_S bzw. K_S' realisierte Algorithmus zur Erzeugung aller Sätze einer Graphsprache, die eine wählbare Knoten- und Kantenzahl nicht überschreiten, beruht im wesentlichen auf einer Aufzählung aller in der nicht-programmierten bzw. programmierten Grammatik vorhandenen Möglichkeiten zur Ableitung von Graph-Satzformen. Hierzu gehören nicht nur alle zulässigen Produktionsfolgen, sondern auch alle durch die jeweilige Satzform gegebenen Möglichkeiten, dieselbe Produktion in verschiedenen Kontextsituationen anzuwenden. Zur Ordnung dieser Vielfalt von Ableitungsmöglichkeiten eignet sich eine Baumstruktur, in der jeder Knoten die Anwendung einer Produktion in einer bestimmten Kontextsituation repräsentiert. Der Baum ist so aufgebaut, daß jeder Weg von der Wurzel zu einem Knoten einer in der gegebenen Grammatik zulässigen Ableitung einer terminalen oder nichtterminalen Graph-Satzform entspricht. Dies hat zur Folge, daß von einem speziellen Wurzelknoten ausgehend Äste zu den verschiedenen Startproduktionen der Grammatik führen. Die Nachfolger jeder Startproduktion sind Kettenproduktionen der Form a) mit entsprechender linker Seite und deren Nachfolger sind ganz allgemein alle in der Grammatik zulässigen Ketten- oder Sternproduktionen, deren linke Seite in der bis zum entsprechenden Knoten abgeleiteten Graph-Satzform enthalten

ist. Eine Vorstellung von dieser Baumstruktur soll das folgende Bild vermitteln:

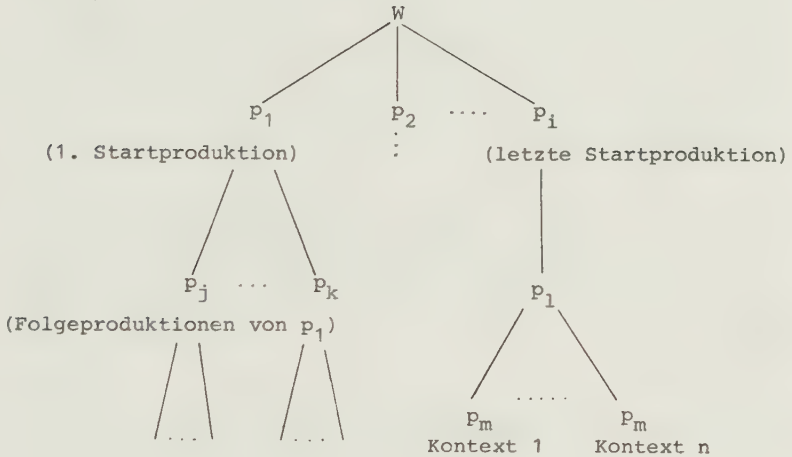


Bild 10

Baumstruktur der Ableitungsmöglichkeiten

Der Aufzählalgorithmus besucht nun alle Knoten, die auf dem Weg zu einem terminalen Graphen mit einer Knoten- und Kantenzahl innerhalb des zulässigen Wertebereichs liegen können, aus Gründen der Speicherplatzökonomie nach der Strategie "depth-first" [De 72]. Bei jedem Schritt auf ein tiefergelegenes Baumniveau wird die entsprechende Produktion auf die Graph-Satzform angewendet und bei jedem Zurückgehen auf ein höhergelegenes Niveau wieder rückgängig gemacht.

In jedem Knoten wird als erstes geprüft, ob die bis dahin erzeugte Graph-Satzform die maximal zulässige Größe überschreitet, und falls dies der Fall ist, wird die zuletzt angewandte Produktion rückgängig gemacht und nach einer anderen Folgeproduktion weitergesucht. Ist die Graph-Satzform nicht zu groß, wird als nächstes festgestellt, ob sie terminal ist. Wenn dies zutrifft, wird der erzeugte Graph gespeichert, gezählt und das Verfahren beim Vorgängerknoten des gerade betrachteten Knotens

fortgesetzt. Ist die Graph-Satzform nichtterminal, so wird überprüft, ob weiterhin die Möglichkeit besteht, aus dieser Satzform einen terminalen Graphen der gewünschten Größe abzuleiten, und falls dies der Fall ist, wird das Verfahren ein Baumniveau tiefer mit einem Nachfolgerknoten weitergeführt. Wenn aber z.B. in der Graph-Satzform mehr als doppelt so viele nichtterminale Knoten wie noch hinzufügbare Kanten o.ä. enthalten sind, woraus sich entnehmen läßt, daß dieser Ableitungsweg zu keinem terminalen Graphen der gewünschten Größe führen kann, so wird im Baum ein Niveau höher gegangen und dort nach einer neuen Alternative gesucht. Das Aufzählverfahren endet nach der Behandlung aller Ableitungsmöglichkeiten für Graphen der Größe kleiner oder gleich X_g mit der Rückkehr zum Wurzelknoten.

Die nach dem beschriebenen Verfahren erzeugten Teilmengen von Graphen aus einer Sprache $L(G)$ haben immer dann, wenn es in G mehr als eine Möglichkeit zur Erzeugung eines bestimmten Graphen X gibt, den großen Nachteil, äquivalente Graphen zu enthalten, so daß die Mächtigkeit der erzeugten Menge im allgemeinen noch nicht zur Berechnung des Gütekriteriums K_g bzw. K_g' herangezogen werden kann.

Die dafür notwendige Reduzierung der Anzahl aller mehrfach vorhandenen äquivalenten Graphen auf jeweils ein Exemplar erfordert einen Algorithmus zur Feststellung der Äquivalenz zweier knoten- und kantenmarkierter gerichteter Graphen, von dessen Effizienz die Praktikabilität der gesamten K_g -Berechnung entscheidend abhängt.

Die in der Literatur beschriebenen Erfahrungen mit dem Problem der Isomorphie von meist ungerichteten und unmarkierten Graphen [CoGo 70, De 72, Le 74] machen deutlich, daß dieses Problem zwar eine triviale Lösung in Form des Ausprobierens aller $n!$ Permutationsmöglichkeiten für n Graphknoten besitzt, diese Lösung aber ab circa zehn Knoten vom Rechenzeitbedarf her gesehen nicht mehr realisierbar ist.

Die erfolgversprechendsten Strategien zur Reduzierung des Aufwands für die Äquivalenz-Prüfung zweier Graphen bestehen zum

einen darin, möglichst viele Äquivalenzinvariante Eigenschaften der Graphen noch vor Beginn des Strukturvergleichs auf Übereinstimmung zu prüfen, um auf diese Weise einen großen Teil der nichtäquivalenten Graphen frühzeitig erkennen zu können, und zum anderen darin, die Anzahl der notwendigen Permutationen zu verkleinern durch eine Partition der Knotenmenge in möglichst viele Teilmengen mit unterschiedlichen Struktureigenschaften wie z.B. Anzahl der ein- und auslaufenden Kanten o.ä.

Das zur Eliminierung mehrfacher Vorkommen von äquivalenten Graphen in der vom Aufzählprogramm gelieferten Menge $L_A(G, X_S)$ erstellte Programm (ca. 400 FORTRAN-Statements) macht von diesen beiden Strategien Gebrauch, indem es zuerst die folgenden Äquivalenzinvarianten Eigenschaften zweier Graphen auf Gleichheit prüft, bevor es mit der Permutation von Knoten beginnt:

- a) Anzahl und Markierungen der Graphknoten müssen übereinstimmen.
- b) Anzahl und Markierungen der Graphkanten müssen übereinstimmen.
- c) Eine Sortierung der Graphknoten in Teilmengen mit den gemeinsamen Merkmalen Knotenmarkierung, Anzahl einlaufender Kanten sowie Anzahl auslaufender Kanten muß gleiche Ergebnisse liefern.
- d) Eine Sortierung der Graphkanten in Teilmengen mit den gemeinsamen Merkmalen Kantenmarkierung, Anzahl ein- bzw. auslaufender Kanten im Quellknoten sowie Anzahl ein- bzw. auslaufender Kanten im Zielknoten muß gleiche Ergebnisse liefern.

Auf den Vergleich von komplizierter zu berechnenden Eigenschaften [Kn 71, Le 74] wurde verzichtet, weil sich anhand der obigen Strukturmerkmale schon ein großer Teil von nicht äquivalenten Graphen gleicher Größe erkennen läßt und es zweifelhaft erscheint, ob es sich lohnt, an dieser Stelle weiteren Aufwand zu investieren.

Falls sich für zwei Graphen bei allen bisher geprüften Eigenschaften keine Unterschiede ergeben haben, werden die Knoten des zweiten Graphen so umnummeriert, daß in beiden Graphen zwei Knoten mit gleicher Nummer auch in bezug auf die Merkmale Knotenmarkierung, Anzahl einlaufender Kanten und Anzahl auslaufender Kanten übereinstimmen. Wenn die beiden Graphen nach dieser Umnummerierung noch nicht identisch sind, dann besteht die einzige Möglichkeit, diesen Zustand herbeizuführen, in der Vertauschung von Knoten, die bei der unter c) beschriebenen Sortierung in die gleiche Teilmenge eingeordnet wurden. Falls bei der Sortierung nach c) k Teilmengen mit jeweils l_i Elementen entstanden sind, so beträgt die Zahl der im ungünstigsten Fall durchzuführenden Permutationen zur Äquivalenzfeststellung

$$AP = \prod_{i=1}^k (l_i!) .$$

Diese Zahl ist im allgemeinen wesentlich kleiner als die Zahl der $n!$ Permutationsmöglichkeiten der Knotenmenge, die bei Nichtberücksichtigung der Knotenmerkmale schlimmstenfalls alle überprüft werden müßten.

Trotz der geschilderten Maßnahmen zur Beschleunigung der Äquivalenzfeststellung liegen die Rechenzeiten für die Komprimierung der Menge $L_A(G, X_S)$ zur für die K_S' -Berechnung notwendigen Menge $L(G, X_S)$ um eine bis mehrere Größenordnungen über den Rechenzeiten des Programms zur Aufzählung der Menge $L_A(G, X_S)$. Dies bedeutet, daß K_S' -Berechnungen nur bis zu Mengen $L_A(G, X_S)$ mit einer Mächtigkeit von einigen hundert Graphen mit vertretbarem Rechenaufwand durchführbar sind. Sobald aber die Anzahl der Graphen in $L_A(G, X_S)$ vierstellig wird, muß man entweder weitergehende Maßnahmen zur Beschleunigung der Äquivalenzfeststellung ergreifen oder auf die exakte Berechnung von K_S' verzichten und die gewünschte Grammatikauswahl mit Hilfe der Mächtigkeit der Mengen $L_A(G_1, X_S)$ und $L_A(G_2, X_S)$ treffen.

Beispiel 37:

Zu vergleichen seien die sechs Grammatiken G_{11} , G_{21} , G_{31} , G_{41} , G_{51} und G_{1010} aus den Beispielen 22 und 25.

Das Gütekriterium K_P liefert für diese Grammatiken die Ergebnisse:

$$K_P(G_{11}) = 5$$

$$K_P(G_{21}) = 7$$

$$K_P(G_{31}) = K_P(G_{41}) = K_P(G_{51}) = K_P(G_{1010}) = 9 .$$

Bei den dazugehörigen programmierten Grammatiken ergeben sich folgende Werte für K_K :

$$K_K(G_{11}) = 0,75$$

$$K_K(G_{21}) = 0,67$$

$$K_K(G_{31}) = K_K(G_{41}) = K_K(G_{51}) = 0,71$$

$$K_K(G_{1010}) = 0,69 .$$

Die K_S -Berechnung ist sowohl für die nichtprogrammierten als auch für die programmierten Grammatiken so aufwendig, daß keines dieser Ergebnisse in weniger als drei Stunden CPU-Zeit zu ermitteln ist.

Ein Versuch, die Verallgemeinerungsfähigkeit der Grammatiken über die Berechnung von Werten für K_S' mit Graphen X_S , die dem ersten und zweiten Stichprobenelement von der Größe her entsprechen, abzuschätzen, liefert für die nichtprogrammierten Grammatiken folgende Ergebnisse:

a) für Graphen X_S mit fünf Knoten und sechs Kanten erhält man:

$$K_S'(G_{11}, X_{5,6}) = 0,028$$

$$K_S'(G_{21}, X_{5,6}) = 0,200$$

$$K_S'(G_{31}, X_{5,6}) = K_S'(G_{41}, X_{5,6}) = K_S'(G_{51}, X_{5,6}) =$$

$$= K_S'(G_{1010}, X_{5,6}) = 0,167$$

b) für Graphen X_S mit sieben Knoten und neun Kanten sind mit akzeptablem Aufwand nur noch berechenbar:

$$K_S'(G_{41}, X_{7,9}) = 0,027$$

$$K_S'(G_{51}, X_{7,9}) = 0,014 .$$

Alle übrigen Werte für K_S' sind mit Sicherheit kleiner als 0,010.

Diese sehr ungünstigen Resultate sind darauf zurückzuführen, daß fast alle Knoten der Stichprobengraphen die gleiche Markierung tragen und dadurch die Zahl der Variationsmöglichkeiten für die Anwendung verschiedener Produktionen in verschiedenen Kontextsituationen sehr rasch ansteigt.

Etwas bessere Ergebnisse liefert die K_S' -Berechnung für die programmierten Grammatiken:

$$a) \quad K_S'(G_{11}, X_{5,6}) = 0,083$$

$$K_S'(G_{21}, X_{5,6}) = 0,200$$

$$\begin{aligned} K_S'(G_{31}, X_{5,6}) &= K_S'(G_{41}, X_{5,6}) = K_S'(G_{51}, X_{5,6}) = \\ &= K_S'(G_{1010}, X_{5,6}) = 0,250 \end{aligned}$$

$$b) \quad K_S'(G_{21}, X_{7,9}) = 0,020$$

$$K_S'(G_{31}, X_{7,9}) = 0,033$$

$$K_S'(G_{41}, X_{7,9}) = 0,047$$

$$K_S'(G_{51}, X_{7,9}) = 0,047$$

$$K_S'(G_{1010}, X_{7,9}) = 0,029$$

Hieraus läßt sich das Fazit ziehen, daß alle für die "Häuschen-Stichprobe" inferierten Grammatiken die Stichprobe außerordentlich stark verallgemeinern und die programmierte Grammatik G_{31} noch die relativ beste unter denjenigen Grammatiken ist, die auch vier- und mehrstöckige "Häuschen" generieren können.

Beispiel 38:

Zu den in Beispiel 24 beschriebenen Stichproben und Grammatiken zur Darstellung der Schaltelemente Widerstand und Diode liefert eine Berechnung der Gütekriterien K_S für die nichtprogrammierten und programmierten Grammatiken folgende Resultate:

a) Für die nichtprogrammierten Grammatiken erhält man

$$K_S(G_W) = 0,086$$

$$K_S(G_D) = 0,421$$

b) Für die programmierten Grammatiken lauten die entsprechenden Ergebnisse:

$$K_S(G_W) = 1$$

$$K_S(G_D) = 1.$$

Dieses Beispiel demonstriert in eindrucksvoller Weise, daß die Programmierung auch ein sehr wirkungsvolles Hilfsmittel zur Verbesserung einer Grammatik im Hinblick auf das Gütekriterium K darstellt.

7.2 STATISTISCHE GÜTEKRITERIEN

Statistische Gütekriterien werden als Maß für den Grad der Übereinstimmung der statistischen Eigenschaften einer inferierten stochastischen Grammatik mit einer gegebenen Experimentmenge benutzt.

Notwendige Grundvoraussetzung für die Durchführbarkeit eines derartigen Vergleichs ist die Konsistenz der inferierten stochastischen Graphgrammatik, weshalb man bei Grammatiken, deren Konsistenz zweifelhaft ist, von der Verwendung statistischer Gütekriterien absehen sollte. Eine weitere Voraussetzung für die bequeme Berechenbarkeit eines statistischen Gütekriteriums während eines Syntaxanalyselaufs der Stichprobe ist die Eindeutigkeit der Grammatik, denn andernfalls lassen sich die Ableitungswahrscheinlichkeiten $q(X_i)$ eines Graphen X_i durch eine stochastische Graphgrammatik G_S nur mit Hilfe eines aufwendigen Spracherzeugungsprogramms (siehe Kapitel 7.1) ermitteln.

Die für eindeutige und konsistente stochastische Grammatiken gebräuchlichen Gütekriterien [Ma 74] beruhen alle auf einem

Vergleich der Ableitungswahrscheinlichkeiten $q(X_i)$ mit den Vorkommenshäufigkeiten c_i dieser Graphen in der Experimentmenge E. Um diese beiden Zahlen aber überhaupt vergleichbar zu machen, muß man die Ableitungswahrscheinlichkeiten $q(X_i)$ mit der Anzahl aller in der Experimentmenge repräsentierten Graphen

$M = \sum_{i=1}^n c_i$ multiplizieren, um einen Erwartungswert für die Vorkommenshäufigkeit der Graphen X_i in einer durch die stochastische Grammatik erzeugten Experimentmenge gleichen Umfangs zu erhalten. Ein Kriterium zur Summation der Unterschiede zwischen den $q(X_i) \cdot M$ und den c_i ist z.B.

$$K_W(G_S) = \sum_{i=1}^n |q(X_i) \cdot M - c_i|.$$

Statt des Absolutbetrags der Differenz könnte man diese z.B. auch quadrieren, wodurch große Unterschiede stärker hervorgehoben würden. In jedem Fall gilt, daß die Experimentmenge umso besser durch die stochastische Grammatik charakterisiert wird, je kleiner der Wert von K_W ausfällt.

Beispiel 39:

Für die eindeutige und konsistente stochastische Graphgrammatik aus Beispiel 34 und 35 läßt die Berechnung des strukturellen Gütekriteriums

$$K_S(G) = 1$$

auf völlige Übereinstimmung von S^+ mit $L(G)$ schließen.

Die Berechnung des statistischen Gütekriteriums

$$\begin{aligned} K_W(G) &= \sum_{i=1}^3 \left| \left(\frac{4}{11}\right)^{i-1} \cdot \frac{7}{11} \cdot 7 - c_i \right| \\ &= |4,45 - 4| + |1,62 - 2| + |0,59 - 1| \\ &= 0,45 + 0,38 + 0,41 \\ &= 1,24 \end{aligned}$$

zeigt jedoch, daß die statistischen Eigenschaften von Ex-

perimentmenge und stochastischer Grammatik noch nicht so gut harmonieren. Dieser Zustand läßt sich aber durch Verwendung einer umfangreicheren Experimentmenge zur Inferenz der stochastischen Grammatik noch deutlich verbessern.

8. VORSCHLÄGE FÜR WEITERE UNTERSUCHUNGEN

Das in dieser Arbeit vorgestellte Konzept zur Inferenz und Syntaxanalyse von Graphgrammatiken für die Bildanalyse kann trotz der sehr unterschiedlichen Strategien zur Lösung des Inferenzproblems als eine Art Weiterentwicklung der Ansätze von Bhargava/Brayer/Fu [BhFu 74, BrFu 77] und Gonzalez/Thomason [GoTh 74, GoTh 78] zur Inferenz von zweidimensionalen Bildbeschreibungsgrammatiken angesehen werden. Die im Vergleich zu Bäumen wesentlich flexibleren Graphstrukturen bringen eine Reihe zusätzlicher Probleme bei der Erzeugung und Analyse von terminalen Graphen mit sich, die aber mit vertretbarem Aufwand zu bewältigen sind. Obwohl es bisher keine Erfahrungen mit Gütekriterien für Baumgrammatiken gibt, ist den publizierten Beispielen für inferierte Baumgrammatiken anzusehen, daß auch sie eine gegebene Stichprobe sehr stark verallgemeinern.

Der vorgestellte Ansatz liefert insbesondere in Anwendungsfällen, die von der Möglichkeit, verschiedene Knotenmarkierungen und verschiedene Kantenmarkierungen zu verwenden, nur wenig Gebrauch machen, noch nicht in jeder Hinsicht zufriedenstellende Ergebnisse. *Verbesserungen* dieses Zustands ließen sich erzielen, wenn es gelänge, die Mehrdeutigkeiten im Ableitungsprozeß zu verringern oder gar zu eliminieren, wodurch die Anzahl der Sackgassen bei der Syntaxanalyse und die Menge der zusätzlich zur Stichprobe erzeugbaren Graphen ganz erheblich reduziert werden könnten.

Die Rechenzeiten für die Inferenz und Syntaxanalyse mit Hilfe von programmierten Graphgrammatiken liegen für Graphen mit weniger als dreißig Knoten in einem durchaus akzeptablen Bereich. Für größere Stichprobenelemente und für die Berechnung des Gütekriteriums K_S kann es jedoch lohnend sein, weiteren Aufwand in die Beschleunigung der beschriebenen Algorithmen zu investieren.

Naheliegende *Verallgemeinerungen* des Ansatzes können z.B. darin

bestehen, wahlweise mit gerichteten oder ungerichteten Kanten arbeiten zu können, wobei für ungerichtete Kanten eine Neufassung der geschilderten Verfahren sinnvoll erscheint.

Erweiterungen des Konzepts auf fehlerkorrigierende Syntaxanalyse und sein Einsatz für Clustering-Verfahren sind grundsätzlich möglich. Eine wesentliche Voraussetzung dafür ist jedoch die Definition eines effizient berechenbaren Abstandsmaßes für Graphen, an dem es bisher noch zu fehlen scheint. Die bisherigen Ansätze in dieser Richtung auf der Basis von kontextfreien Zeichenkettengrammatiken [Fu 77b] und Baumgrammatiken [LuFu 78] arbeiten mit Produktionen als Formalisierungshilfsmittel für die Entstehung bzw. Beseitigung eines Fehlers, wodurch sich die Anzahl der Produktionen in der Grammatik und die Syntaxanalysezeiten vervielfachen. Nachdem sich derartige Effekte bei Graphgrammatiken noch in weitaus stärkerem Maße bemerkbar machen, wäre es höchst interessant zu untersuchen, um wieviel besser sich diese Aufgabenstellung durch ein Syntaxanalyseverfahren, das Graphen mit einem vorgebbaren Maximalabstand zu einem durch die Grammatik erzeugbaren Graphen akzeptiert, bewältigt werden kann. Der Aufwand für die Realisierung eines derartigen Verfahrens ist mit Sicherheit größer als der für eine gewöhnliche Syntaxanalyse wie in Kapitel 4, brächte aber bei der Erstellung der Stichprobe den nicht zu unterschätzenden Vorteil mit sich, daß man nur eine Stichprobe aus Prototypen statt einer Stichprobe, die alle Arten von Fehlern in allen denkbaren Varianten enthält, zu verarbeiten bräuchte.

Von der Idee her äußerst attraktiv sind auch mehrstufige Graphgrammatiken zur Analyse von hierarchisch aufgebauten Mustern [Ni 80b]. Allerdings fehlt es hierfür noch am theoretischen Unterbau wie z.B. an Ergebnissen zur Entscheidbarkeit des Graphproblems in mehrstufigen Graphgrammatiken [Gö 77], so daß bis zur Inferenz von mehrstufigen Graphgrammatiken noch ein weiterer Weg zurückzulegen sein wird.

ZUSAMMENFASSUNG

Die vorliegende Arbeit beschäftigt sich mit dem Problem der Inferenz und Syntaxanalyse von Graphgrammatiken für die syntaktische Mustererkennung.

Nach einer einführenden Darstellung der Grundvoraussetzungen für den syntaktischen Ansatz in der Mustererkennung folgt eine Übersicht über die bekanntesten ein- und zweidimensionalen Bildbeschreibungsfomalismen. Eine Zusammenstellung der wichtigsten enumerativen und konstruktiven Inferenzverfahren zeigt, daß der Stand der Kunst gerade bei Bildbeschreibungssprachen, die sich für die Darstellung komplexer zweidimensionaler Linienmuster eignen, noch sehr zu wünschen übrig läßt.

Vor diesem Hintergrund wird eine spezielle Klasse von Graphgrammatiken vorgeschlagen, mit der gerichtete, knoten- und kantenmarkierte Graphen zur Beschreibung von beliebig strukturierten Linienmustern erzeugbar sind. Die Produktionen dieser kontextsensitiven Graphgrammatiken generieren Teilgraphen, die die Form von gerichteten Kantenzügen bzw. Büscheln von Kanten mit gemeinsamen Ausgangsknoten besitzen und deshalb als Ketten- bzw. Sternproduktionen bezeichnet werden.

Für die Inferenz von Graphgrammatiken dieses Typs werden zwei verschiedene Inferenzverfahren vorgestellt, die sich beide einer konstruktiven und inkrementellen Vorgehensweise bedienen. Das Inferenzverfahren ohne Gedächtnis arbeitet nach der Strategie, jeden Graphen der Stichprobe aus möglichst wenigen Ketten- und Sternproduktionen aufzubauen und die so erhaltenen Produktionen zu einer Gesamtgrammatik zu vereinigen. Das Inferenzverfahren mit Gedächtnis hingegen ist bestrebt, die Zahl der Produktionen in der Gesamtgrammatik niedrig zu halten, indem es bereits früher konstruierte Produktionen zum Aufbau neuer Stichprobenelemente anzuwenden versucht. Bei beiden Inferenzverfahren kann die maximale Größe der durch eine Produktion erzeugbaren Ketten bzw. Sterne über die Eingabe von zwei Parametern gesteuert werden.

Die Experimente mit beiden Inferenzverfahren zeigen, daß die Verarbeitung eines Stichprobenelements i.a. nur wenige Sekunden Rechenzeit in Anspruch nimmt und somit auch Grammatiken für große Stichproben mit vertretbarem Aufwand inferiert werden können. Die Unterschiede zwischen beiden Inferenzverfahren äußern sich im wesentlichen darin, daß der Erhalt einer brauchbaren Grammatik beim Inferenzverfahren ohne Gedächtnis stärker von der Wahl geeigneter Parameter und beim Inferenzverfahren mit Gedächtnis von der Verarbeitungsreihenfolge der Stichprobenelemente beeinflußt wird.

Um die vom Inferenzverfahren gelieferten Graphgrammatiken zur Klassifikation weiterer Muster desselben Problemkreises einsetzen zu können, wird ein Syntaxanalyseverfahren für Graphgrammatiken mit Ketten- und Sternproduktionen entwickelt. Das Verfahren arbeitet nichtdeterministisch, top-down und versucht die Strategie der Inferenzverfahren genau nachzuvollziehen. Der Aufwand für die Syntaxanalyse eines Graphen ist in erster Linie abhängig von der Anzahl der Möglichkeiten, beim Ableitungsprozeß in eine Sackgasse zu geraten, und steigt für manche Grammatiken mit der Länge der Ableitung außerordentlich rasch an. Die Rechenzeiten für die Syntaxanalyse eines Graphen sind deshalb sehr unterschiedlich und liegen i.a. im Bereich von wenigen Sekunden bis einigen Minuten.

Als sehr wirkungsvolles Hilfsmittel sowohl zur Einschränkung des Sprachumfangs inferierter Grammatiken als auch zur Reduzierung des Aufwands für die Syntaxanalyse erweist sich die Verwendung programmierter Graphgrammatiken. Die Programmierung einer Grammatik ist aus den Produktionenfolgen zur Erzeugung der Stichprobenelemente leicht ermittelbar und trägt durch das Ausschalten von Sackgassen ganz erheblich zur Beschleunigung der Syntaxanalyse bei.

Erweiterungen des Ansatzes auf stochastische Graphgrammatiken und stochastische Syntaxanalyse sind nur dann sinnvoll, wenn das Inferenzverfahren eine eindeutige und konsistente Grammatik geliefert hat. Dies ist insbesondere bei umfangreicheren Grammatiken jedoch meist nicht der Fall, so daß sich die geschätzten Wahrscheinlichkeiten höchstens zur Sortierung von Alternativen nach fallender

Anwendungshäufigkeit verwenden lassen.

Zum Problem der Auswahl einer im Hinblick auf eine bestimmte Anwendung geeignetsten Graphgrammatik werden Gütekriterien vorgeschlagen, die mit der Komplexität und der Verallgemeinerungsfähigkeit von Graphgrammatiken die beiden wichtigsten Eigenschaften inferierter Grammatiken erfassen.

Zum Abschluß der Arbeit wird auf weitere Verbesserungs-, Verallgemeinerungs- und Erweiterungsmöglichkeiten des vorgestellten Ansatzes eingegangen.

SCHLUSSBEMERKUNG

An dieser Stelle möchte ich mich bei Herrn Prof. Dr. H. Niemann für die Betreuung der Arbeit und bei Herrn Prof. Dr. H. J. Schneider für die Übernahme des Zweitgutachtens bedanken.

Mein Dank gilt auch Herrn Prof. Dr. M. Nagl sowie meinen Kollegen H. Bley, Dr. H. Bunke, H.-W. Hein und D. Jäpel für ihre Diskussionsbereitschaft und nicht zuletzt Frau I. Franken für die sorgfältige Reinschrift des Manuskriptes.

LITERATURVERZEICHNIS

- [AbMiToTa 73] ABE,N.; MIZUMOTO,M.; TOYODA,J.-I.; TANAKA,K.: Web Grammars and Several Graphs, Journal of Computer and System Sciences 7 (1973), S. 37-65
- [AhUl 72] AHO,A.V.; ULLMAN,J.D.: The Theory of Parsing, Translation and Compiling, Volume 1: Parsing, Prentice-Hall, Englewood Cliffs 1972
- [Ba 78] BARTSCH,B.: Inferenz von Grammatiken für die Bildanalyse, Interner Bericht, IMMD Erlangen 1978
- [Bh 74] BHARGAVA,B.K.: Tree Systems for Syntactic Pattern Recognition, Ph.D. Thesis, Purdue University 1974
- [BhFu 74] BHARGAVA,B.K.; FU,K.S.: Transformations and Inference of Tree Grammars for Syntactic Pattern Recognition, Proceedings IEEE Int. Conference on Systems, Man and Cybernetics, Dallas 1974, S. 330-333
- [BiFe 70] BIERMANN,A.W.; FELDMAN,J.A.: On the Synthesis of Finite-State Acceptors, Technical Report AIM-114, Stanford University 1970
- [BiFe 72] BIERMANN,A.W.; FELDMAN,J.A.: A Survey of Results in Grammatical Inference, in Watanabe,S. (ed.): Frontiers of Pattern Recognition, Academic Press, New York 1972, S. 31-54
- [Bl 80] BLEY,H.: Bildgraphen für die Segmentierung von Stromlaufplänen, Tagungsband des 3. DAGM-Symposiums, Essen 1980
- [BoTh 73] BOOTH,T.L.; THOMPSON,R.A.: Applying Probability Measures to Abstract Languages, IEEE Transactions on Computers, Vol. C-22 (1973), S. 442-450
- [Br 69] BRAINERD,W.S.: Tree Generating Regular Systems, Information and Control 14 (1969), S. 217-231
- [Br 77] BRAYER,J.M.: Parsing of Web Grammars, Proceedings IEEE-Workshop on Data Description and Management, Long Beach 1977, S. 102
- [BrFu 75] BRAYER,J.M.; FU,K.S.: Web Grammars and their Application to Pattern Recognition, Technical Report TR-EE 75-1, Purdue University, West Lafayette 1975

- [BrFu 77] BRAYER, J.M.; FU, K.S.: A Note on the k-Tail Method of Tree Grammar Inference, IEEE Transactions on Systems, Man and Cybernetics 7 (1977), S. 293-300
- [Bu 78] BUNKE, H.: Analyse elektrischer Schaltpläne mit einfachen Schaltsymbolen, in Triendl, E. (ed.): 1. DAGM-Symposium "Bildverarbeitung und Mustererkennung", Informatik-Fachberichte Band 17, Springer Verlag Berlin 1978, S. 126-132
- [Bu 79] BUNKE, H.: Sequentielle und parallele programmierte Graph-Grammatiken, Dissertation, Universität Erlangen-Nürnberg 1979
- [Ch 59] CHOMSKY, N.: On certain formal properties of grammars, Information and Control 2 (1959), S. 137-167
- [Ch 64] CHOMSKY, N.: Aspects of the theory of syntax, M.I.T. Press, Cambridge 1964
- [ChFu 76] CHOU, S.M.; FU, K.S.: Inference for Transition Network Grammars, Proceedings 3rd Int. Joint Conference on Pattern Recognition, Coronado 1976, S. 79-84
- [CoGo 70] CORNEIL, D.C.; GOTLIEB, C.C.: An Efficient Algorithm for Graph Isomorphism, Journal of the ACM, vol. 17 (1970), S. 51-64
- [CoRo 74] COOK, C.M.; ROSENFELD, A.: Some Experiments in Grammatical Inference, in Simon, J.C. (ed.): Computer-oriented learning processes, Preprints NATO-ASI Workshop, Bonas 1974
- [CoRoAr 76] COOK, C.M.; ROSENFELD, A.; ARONSON, A.R.: Grammatical Inference by Hill Climbing, Informational Sciences 10 (1976), S. 59-80
- [Cr 70] CRESPI-REGHIZZI, S.: The Mechanical Acquisition of Precedence Grammars, Dissertation, University of California, Los Angeles 1970
- [Cr 71a] CRESPI-REGHIZZI, S.: An effective model for grammar inference, Proceedings IFIP-Congress Ljubljana 1971, S. 524-529
- [Cr 71b] CRESPI-REGHIZZI, S.: Reduction of Enumeration in grammar acquisition, Proceedings 2nd Int. Joint Conference on Artificial Intelligence London 1971, S. 546-552

- [De 74] DEO,N.: Graph Theory with Applications to Engineering and Computer Science, Prentice Hall, Englewood Cliffs 1974
- [Do 70] DONER,J.: Tree Acceptors and Some of Their Applications, Journal of Computer and System Sciences 4 (1970), S. 406-451
- [Du 79] DULZ,W.: Konstruktive Verfahren zur Gewinnung von Grammatiken für die Musteranalyse, Diplomarbeit, IMMD Erlangen 1979
- [El 79] ELLERMANN,R.: Enumerative Verfahren zur Gewinnung von Grammatiken für die Musteranalyse, Studienarbeit, IMMD Erlangen 1979
- [Ev 71] EVANS,T.G.: Grammatical Inference Techniques in Pattern Analysis, in Tou,J.T. (ed.): Software Engineering vol. 2, Academic Press, New York 1971, S. 183-202
- [Fe 71] FEDER,J.: Plex Languages, Information Sciences 3 (1971), S. 225-241
- [Fe 72] FELDMAN,J.: Some Decidability Results on Grammatical Inference and Complexity, Information and Control 20 (1972), S. 244-262
- [FeGiHoRe 69] FELDMAN,J.A.; GIPS,J.; HORNING,J.J.; REDER,S.: Grammatical Complexity and Inference, Technical Report TR-CS 125, Stanford University 1969
- [Fr 76] FRANCK,R.: PLAN2D-Syntactic Analysis of Precedence Graph Grammars, Proceedings 3rd ACM Symposium on Principles of Programming Languages, Atlanta 1976, S. 134-139
- [Fu 72] FU,K.S.: On Syntactic Pattern Recognition and Stochastic Languages, in Watanabe,S. (ed.): Frontiers of Pattern Recognition, Academic Press, New York 1972, S. 113-137
- [Fu 74] FU,K.S.: Syntactic Methods in Pattern Recognition, Academic Press, New York 1974
- [Fu 76a] FU,K.S.: Digital Pattern Recognition, Springer Verlag, Berlin 1976
- [Fu 76b] FU,K.S.: Tree Languages and Syntactic Pattern Recognition, in Chen,C.H. (ed.): Pattern Recognition and Artificial Intelligence, Academic Press, New York 1976
- [Fu 77a] FU,K.S.: Syntactic Pattern Recognition, Applications, Springer Verlag Berlin 1977

- [Fu 77b] FU,K.S.: Error-Correcting Parsing for Syntactic Pattern Recognition, in Klinger,A.; Fu,K. S.; Kunii,T.L. (eds.): Data Structures, Computer Graphics and Pattern Recognition, Academic Press, New York 1977
- [Fu 78] FU,K.S.: Recent Advances in Syntactic Pattern Recognition, 4th IJCPR Kyoto 1978, S. 13-18
- [FuBh 73] FU,K.S.; BHARGAVA,B.T.: Tree Systems for Syntactic Pattern Recognition, IEEE Transactions on Computers 22 (1973), S. 1087-1099
- [FuBo 75] FU,K.S.; BOOTH,T.L.: Grammatical Inference: Introduction and Survey - Part I and II, IEEE Transactions on Systems, Man and Cybernetics 5, S. 95-111 und 409-423
- [GiSp 68] GINSBURG,S.; SPANIER,E.: Control Sets on Grammars, Math. Syst. Theory 2 (1968) S.159-177
- [Go 67] GOLD,E.M.: Language Identification in the Limit, Information and Control 10 (1967), S. 447-474
- [Gö 77] GÖTTLER,H.: Zweistufige Graphmanipulationssysteme für die Semantik von Programmiersprachen, Dissertation IMMD Erlangen 1977
- [GoTh 74] GONZALEZ,R.C.; THOMASON,M.G.: On the Inference of Tree Grammars for Syntactic Pattern Recognition, Proc. Conference on Systems, Man and Cybernetics, Dallas 1974, S. 311-315
- [GoTh 78] GONZALEZ,R.C.; THOMASON,M.G.: Syntactic Pattern Recognition, Addison-Wesley Publishing Company, Reading 1978
- [Ho 69] HORNING,J.J.: A Study of Grammatical Inference, Technical Report TR-CS 139, Stanford University 1969
- [Ho 80] HOFFMANN,A.: Inferenz von Baumgrammatiken für die Musteranalyse, Studienarbeit, IMMD Erlangen 1980
- [HoUl 69] HOPCROFT,J.E.; ULLMAN,J.D.: Formal Languages and their Relation to Automata, Addison-Wesley, Reading 1969
- [Hu 75] HUNT,E.B.: Artificial Intelligence, Academic Press, London 1975, S. 144-182

- [HuFu 72] HUANG,T.; FU,K.S.: Stochastic Syntactic Analysis for Programmed Grammars and Syntactic Pattern Recognition, Computer Graphics and Image Processing 1 (1972), S. 257-283
- [Ka 74] KANAL,L.: Patterns in Pattern Recognition: 1968-1974, IEEE Transactions on Information Theory 20 (1974), S. 697-722
- [Kn 71] KNÖDEL,W.: Ein Verfahren zur Feststellung der Isomorphie von endlichen, zusammenhängenden Graphen, Computing 8 (1971), S. 329-334
- [KnKn 76] KNOBE,B.; KNOBE,K.: A Method for Inferring Context-free Grammars, Information and Control 31 (1976), S. 126-146
- [Kr 65] KREYSZIG,E.: Statistische Methoden und ihre Anwendungen, Vandenhoeck & Ruprecht, Göttingen 1965
- [Le 65] LEDLEY,R.S. et al.: FIDAC: Film Input to Digital Automatic Computer and Associated Syntax-Directed Pattern-Recognition Programming System, in Tippet,J.T. et al. (eds.): Optical and Electro-Optical Information Processing, M.I.T. Press, Cambridge 1965, S. 591-613
- [Le 74] LEVI,G.: Graph Isomorphism: A Heuristic Edge-Partitioning-Oriented Algorithm, Computing 12 (1974), S. 291-313
- [LeFu 72] LEE,H.C.; FU,K.S.: A Stochastic Syntax Analysis Procedure and Its Application to Pattern Classification, IEEE Transactions on Computers, vol. C-21 (1972), S. 660-666
- [LeFu 74] LEE,H.C.; FU,K.S.: A Syntactic Pattern Recognition System with Learning Capability, in Tou,J.T. (ed.): Information Systems, Plenum Press, New York 1974, S. 425-449
- [LeYu 78] LEVY,L.S.; YUEH,K.: On Labelled Graph Grammars, Computing 20 (1978), S. 109-125
- [Lo 72] LOECKX,J.: Computability and Decidability, Springer Verlag, Berlin 1972
- [LuFu 78] LU,S.Y.; FU,K.S.: Error-Correcting Tree Automata for Syntactic Pattern Recognition, IEEE Transactions on Computers 27 (1978), S. 1040-1053

- [LuFu 79] LU,S.Y.; FU,K.S.: Stochastic Tree Grammar Inference for Texture Synthesis and Discrimination, Computer Graphics and Image Processing 9 (1979), S. 234-245
- [Ma 74] MARYANSKI,F.J.: Inference of Probabilistic Grammars, Dissertation, University of Connecticut, Storrs 1974
- [MaBo 77] MARYANSKI,F.J.; BOOTH,T.L.: Inference of Finite-State Probabilistic Grammars, IEEE Transactions on Computers 26 (1977), S. 521-536
- [Mi 75] MILGRAM,D.L.: Web Automata, Information and Control 29 (1975), S. 162-184
- [Mi 76] MICLET,L.: Inference of regular expressions, Proceedings 3rd Int. Joint Conference on Pattern Recognition, Coronado 1976, S. 100-105
- [Mi 78] MICLET,L.: Multiple Inference in Regular Grammars, Proceedings 4th Int. Joint Conference on Pattern Recognition, Kyoto 1978, S. 341-345
- [Mo 70] MONTANARI,U.G.: Separable Graphs, Planar Graphs and Web Grammars, Information and Control 16 (1970), S. 243-267
- [MoFu 76] MOAYER,B.; FU,K.S.: A Tree System Approach to Fingerprint Pattern Recognition, IEEE Transactions on Computers 25 (1976), S. 262-274
- [MuWa 78] VAN DER MUDE,A.; WALKER,A.: On the Inference of Stochastic Regular Grammars, Information and Control 38 (1978), S. 310-329
- [My 72] MYLOPOULOS,J.: On the Relation of Graph Grammars and Graph Automata, Proceedings 13th Symposium on Switching and Automata Theory (1972), S. 108-120
- [Na 76] NAGL,M.: Formal Languages of Labelled Graphs, Computing 16 (1976), S. 113-137
- [Na 79] NAGL,M.: Graph-Grammatiken: Theorie, Anwendungen, Implementierung, Vieweg Verlag, Braunschweig 1979
- [Ni 74] NIEMANN,H.: Methoden der Mustererkennung, Akademische Verlagsgesellschaft, Frankfurt 1974
- [Ni 80a] NIEMANN,H.: Anwendungen der Mustererkennung, Informatik-Spektrum 3 (1980), S. 19-30

- [Ni 80b] NIEMANN, H.: Zur Repräsentation von Kontrollstrukturen und von Wissen in der Musteranalyse, Tagungsband des 3. DAGM-Symposiums, Essen 1980
- [Pa 72] PAVLIDIS, T.: Linear and Context-Free Graph Grammars, Journal of the ACM, vol. 19 (1972), S. 11-22
- [Pa 77] PAVLIDIS, T.: Structural Pattern Recognition, Springer Verlag Berlin 1977
- [PaAl 79] PAVLIDIS, T.; ALI, F.: A Hierarchical Syntactic Shape Analyzer, IEEE Transactions on Pattern Analysis and Machine Intelligence 1 (1979), S. 1-9
- [Pf 72] PFALTZ, J.L.: Web Grammars and Picture Description, Computer Graphics and Image Processing 1 (1972), S. 193-220
- [PfRo 69] PFALTZ, J.L.; ROSENFELD, A.: Web Grammars, Proceedings Int. Joint Conference on Artificial Intelligence, Washington 1969, S. 609-619
- [Po 76] PORTER, G.B.: Grammatical Inference based on Pattern Recognition, 3rd Int. Joint Conference on Pattern Recognition, Coronado 1976, S. 90-93
- [Ra 73] RAJLICH, V.: Relational Structures and Dynamics of Certain Discrete Systems, Proceedings Conference Mathematical Foundations of Computer Science, High Tatras 1973, S. 285-292
- [RoSn 72] ROSENFELD, A.; SNIVELY, J.W.: Bounds on the Complexity of Grammars, in Watanabe, S. (ed.): Frontiers of Pattern Recognition, Academic Press, New York 1972, S. 479-489
- [Sc 75] SCHNEIDER, H.J.: Compiler - Aufbau und Wirkungsweise, de Gruyter Verlag, Berlin 1975
- [ScFi 79] SCHNUPP, P.; FLOYD, C.: Software: Programmentwicklung und Projektorganisation, de Gruyter, Berlin 1979
- [Sh 69] SHAW, A.C.: A Formal Picture Description Scheme as a Basis for Picture Processing Systems, Information and Control 14 (1969), S. 9-52
- [Sh 78] SHAPIRO, L.G.: Data Structures for Picture Processing, Computer Graphics 12 (1978), S. 140-145

- [So 64] SOLOMONOFF, R.J.: A Formal Theory of Inductive Inference, Part I and II, Information and Control 7 (1964), S. 1-22 und 224-254
- [Ti 76] TINHOFFER, G.: Methoden der angewandten Graphentheorie, Springer-Verlag Wien 1976
- [ViGh 78] DELLA VIGNA, P.; GHEZZI, C.: Context-Free Graph Grammars, Information and Control 37 (1978), S. 207-233
- [Wi 70] WINSTON, P.H.: Learning Structural Descriptions from Examples, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge 1970
- [Wi 78] WINKLER, G.: Bildbeschreibungssprachen - was sie sind und was sie leisten, in Triendl, E. (ed.): 1. DAGM-Symposium "Bildverarbeitung und Mustererkennung", Informatik-Fachberichte Band 17, Springer Verlag Berlin 1978, S. 107-125
- [Wh 77] WHARTON, R.M.: Grammar Enumeration and Inference, Information and Control 33 (1977), S. 253-272

LEBENS LAUF

Geburtstag und -ort:	15.3.1952, Nürnberg
Eltern:	Hubert Bartsch Anna Bartsch, geb. Derrer
1958 - 1962 :	Besuch der Volksschule in Nürnberg
1962 - 1971 :	Besuch des Labenwolf-Gymnasiums in Nürnberg Schulabschluß: Abitur
1971 - 1976 :	Studium an der Universität Erlangen Fachrichtung: Informatik Studienabschluß: Diplom
Seit 1.7.1976 :	Verwalter der Dienstgeschäfte eines wissenschaftlichen Assistenten am Lehrstuhl für Informatik 5 (Mustererkennung) der Universität Erlangen-Nürnberg

In letzter Zeit sind folgende Arbeitsberichte des Institutes für
Mathematische Maschinen und Datenverarbeitung erschienen:

Band 11

- =====
- Nr. 1 Körber, K., Werzinger, G.:
 Das Modell SIM-LAB.
 Beschreibung des Grundmodells.
 (Februar 1978)
- Nr. 2 Körber, K., Werzinger, G.:
 Das Modell SIM-LAB.
 Beschreibung der Modellerweiterung und der Strategien.
 (Februar 1978)
- Nr. 3 Körber, K., Werzinger, G.:
 Das Modell SIM-LAB.
 Listing der Quellprogramme.
 (Februar 1978)
- Nr. 4 Beth, Th.:
 On Resolutions of Steiner Systems.
 (März 1978)
- Nr. 5 Leeb, K.:
 Salami-Taktik beim Quader-Packen.
 Riess, W.:
 Zwei Optimierungsprobleme auf Ordnungen.
 (April 1978)
- Nr. 6 Händler, W., Henning, W., Kleinöder, W., Kneissl, J.,
 Volkert, J.:
 EPIK - Erlanger Projekt Interaktive Kartographie.
 (April 1978)
- Nr. 7 Schreiber, H.:
 Hardware-Messung und Analyse des Ablaufgeschehens in
 Rechnerkernen
 (April 1978)

- =====
- Nr. 8 Weber, D.:
Datengraphen und deren Transformation:
Ein Konzept zur Spezifikation von Datentypen.
(Juni 1978)
- Nr. 9 Musielak, H., Schmidt, B., Stössel, M.:
Vergleich von Simulationssprachen:
GPSS-FORTRAN - GPSS - GASP IV.
(Juli 1978)
- Nr. 10 Solymosi, A.:
Synthese von analysierenden Automaten auf Grund von
formalen Grammatiken.
(Juli 1978)
- Nr. 11 Meyer, K., Schaller, K., Schmidt, B.:
SIM-QUEUE.
Beschreibung eines Simulationsmodells für ein kom-
plexes Warteschlangensystem, das mit Hilfe von Eingabe-
daten zusammengestellt werden kann.
(Juli 1978)
- Nr. 12 Bodendorf, F., Eckardt, T., Mertens, P., Schrammel, D.:
Benutzung und Beurteilung von Kleinrechnersystemen in einer
Hochschule - Ein Beitrag zur Dezentralisierungsdiskussion
(August 1978)
- Nr. 13 Ravinovitz, M.:
An Approach to a Reconfigurable Multiprocessor System.
Bode, A., Händler, W.:
Classification d'architectures parallèles:
Introduction de la notation ECS et application au
projet EGPA
Sigmund, W.:
Parallel Machine Functions in Program Transformation
and Execution
(September 1978)
- Nr. 14 Beth, Th., Strehl, V.:
Materialien zur Codierungstheorie
(September 1978)

- =====
- Nr. 15 Wendler, K.:
Betriebssystemaspekte in hierarchisch modularen Poly-
prozessorsystemen - Modellierungsansätze und Koordinie-
rungsmechanismen
(Oktober 1978)
- Nr. 16 Altmann, W.:
Beschreibung von Programmoduln zum Entwurf zuverlässiger
Softwaresysteme
(Oktober 1978)
- Nr. 17 Hofmann, W.:
Warteschlangenmodelle für die Parallelverarbeitung
(November 1978)
- Nr. 18 Stöth, G.:
Höhere Programmiersprachen für Kleinrechner:
Anforderungen, Entwicklungskriterien, Compileraspekte
(November 1978)
- Nr. 19 Bode, A., Händler, W.:
Rechnerarchitektur - Grundlagen und Verfahren
(November 1978)
- Nr. 20 Kneissl, F.:
Grundsoftware für die Textverarbeitung
(Dezember 1978)
- Nr. 21 Leidel, W.:
Synchronisationsprobleme in Automatenetzen
(Dezember 1978)
- Nr. 22 Bodendorf, F., Renninger, W., Schaller, K.:
Zwei neue Methodenbanksysteme für statistische Anwendungen
(Dezember 1978)
- Nr. 23 Jahresbericht 1978 der Informatik
(März 1979)

- Nr. 1 Elzer, P.:
Strukturierte Beschreibung von Prozesssystemen
(Februar 1979)
- Nr. 2 Wunderatsch, H.:
Deterministisches Zuordnen in heterogenen Prozessor-
systemen mit Pseudo-Boole Methoden
(Juni 1979)
- Nr. 3 Bunke, H.:
Sequentielle und parallele programmierte Graph-Grammatiken
(Juni 1979)
- Nr. 4 Luft, A.L.:
Zur ingenieurwissenschaftlichen Theorie von Rechner-
systemen
(August 1979)
- Nr. 5 Schmidt, B.:
Das Simulationsmodell SIM 330 - Einführung
(August 1979)
- Nr. 6 Gardill, R.:
SKOP, Syntaxnotation als Kontrollstruktur für
Programmiersprachen
(September 1979)
- Nr. 7 Keramidis, S.:
Entwurf und Implementierung eines Rechnerverbund-
netzes für die Fertigungssteuerung.
Fallstudie zur Kontrolle zuverlässiger Systeme
(September 1979)
- Nr. 8 Widjaja, H.:
Die optimale Zerlegung von Netzen und die Optimierung
des Prozeßverhaltens im Adressraum
(Oktober 1979)
- Nr. 9 Keramidis, S., Grothe, W.:
Beiträge zur Lösung des Verklemmungsproblems in prioritäts-
freien Betriebsmittelmaschinen
(Oktober 1979)
- Nr. 10 Beth, Th., Hain, M., Sagerer, G. Schäfer, N.:
Materialien zur Codierungstheorie II (Dezember 1979)
- Nr. 11 Jahresbericht 1979 der Informatik (März 1980)

- Nr. 1 Schrammel, D., Bayer, W., Gliemann, H.-P., Heigl, M.,
Mertens, P.:
Ein Leistungs- und Wirtschaftlichkeitsvergleich zwischen
Klein- und Großcomputern
(Januar 1980)
- Nr. 2 Göttler, H.:
Semantische Modelle
(März 1980)
- Nr. 3 Fromm, H.J.:
Zur Modellierung der Speicherinterferenz bei hierarchisch
organisierten Multiprozessorsystemen
(April 1980)
- Nr. 4 Jäpel, D.:
Klassifikatorbezogene Merkmalsauswahl
(Juli 1980)
- Nr. 5 Bodendorf, F., Mertens, P., Haas, R., Weber, G. Wolf, G.:
SAMBA - ein Methodenbankrahmen um das Statistik-Paket SPSS
(Juli 1980)







